AUTONOMOUS SATELLITE OPERATIONS FOR CUBESAT SATELLITES

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jason L. Anderson

March 2010

© 2010

Jason L. Anderson

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Autonomous Satellite Operations For CubeSat Satellites
AUTHOR:	Jason L. Anderson
DATE SUBMITTED:	March 2010
COMMITTEE CHAIR:	Dr. Franz Kurfess Computer Science Professor Computer Science Department California Polytechnic State University
COMMITTEE MEMBER:	Dr. Jordi Puig-Suari Aerospace Professor Aerospace Department California Polytechnic State University
COMMITTEE MEMBER:	Dr. Alex Dekhtyar Computer Science Associate Professor Computer Science Department California Polytechnic State University

Abstract

Autonomous Satellite Operations For CubeSat Satellites

by

Jason L. Anderson

In the world of educational satellites, student teams manually conduct operations daily, sending commands and collecting downlinked data. Educational satellites typically travel in a Low Earth Orbit allowing line of sight communication for approximately thirty minutes each day. This is manageable for student teams as the required manpower is minimal. The international Global Educational Network for Satellite Operations (GENSO), however, promises satellite contact upwards of sixteen hours per day by connecting earth stations all over the world through the Internet. This dramatic increase in satellite communication time is unreasonable for student teams to conduct manual operations and alternatives must be explored. This thesis first introduces a framework for developing different Artificial Intelligences to conduct autonomous satellite operations for CubeSat satellites. Three different implementations are then compared using Cal Poly's CP6 CubeSat and the University of Tokyo's XI-IV CubeSat to determine which method is most effective.

Keywords: Autonomous Operations, CubeSat, Lights Out Operations, Earth Station, Validation Framework, Rule Based System, Process Extraction

Acknowledgements

There are so many people who have helped to make this thesis possible.

Dr. Franz Kurfess For inspiring my passion for autonomous systems and encouraging your students to be proud and submit their work to public forums.

Dr. Jordi Puig-Suari For all your support and creating the CubeSat project which has made learning an enjoyable/real world experience. Who'd have thought a $10cm^3$ cube could offer so many opportunities?

Dr. Alex Dekhtyar For introducing me to the world of Data Mining and all of its many interesting problems.

PolySat & CubeSat To those who have come before me, thank you for the ground work you have laid to get us to where we are today. To those who have worked beside me, thank you for always being supportive and a friend. To those who are to come, I encourage you to seize this opportunity to learn and challenge yourself.

My Family For always believing in me and encouraging me to follow my dreams.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are many different operational satellites in orbit at the moment with missions ranging from scientific payloads provided by NASA's Jet Propulsion Laboratory (JPL) [28] to commercial communication missions such as Direct TV services [8]. Each of these satellites, however, requires an operations team to monitor the spacecraft and solve potential problems. These operations are well understood, repetitive tasks making spacecraft operations a perfect candidate for automation [17].

Satellite operations can also be expensive to maintain for any sustained period of time. NASA operation centers are typically staffed 24 hours a day, 7 days a week which can add up over time [17]. For instance, LandSat 7 requires approximately $20 million per year for operations [45]. If automation made it possible to reduce this budget by even 5% ($1 million), the direct savings alone would be enough to adopt an automated system.

In order to provide students with the necessary skills to work in the Aerospace industry, Stanford University in coordination with the California Polytechnic

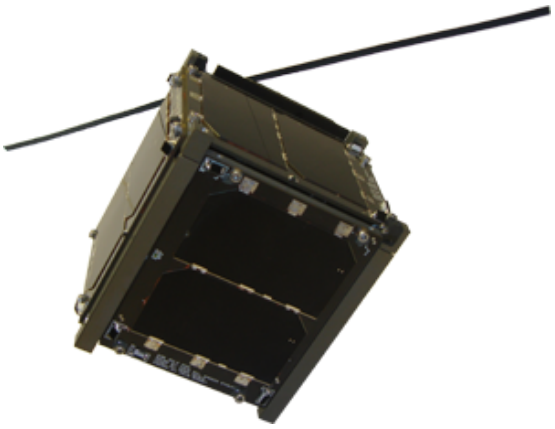


Figure 1.1: CP6, Cal Poly's Forth CubeSat

State University (Cal Poly) has developed the CubeSat standard [3]. CubeSats are small $10cm^3$ satellites weighing less than a kilogram (see Figure 1.1 for an image of Cal Poly's CP6 CubeSat) [46]. The idea is that small satellites can be developed in approximately 2 years, allowing students to be involved in the design, development, testing and operations of a complete spacecraft. There are currently over 20 CubeSats in orbit at various mission stages [24] (see Figure 1.5 for an image of CP4 in space taken by Aerospace Corporation's AeroCube-2).

Figure 1.2: Cal Poly's Two Earth Stations

Unlike NASA missions which typically have 24-hour contact with their spacecraft, CubeSat orbits are such that only 30 minutes of contact is available per day [20]. Additionally, the commands that CubeSat operators send are simplistic such as taking a picture or dumping onboard data. These commands do not require any complex sequencing. CubeSat operations therefore require a small amount of manpower and currently do not warrant a fully automated system as CubeSat teams often have 10 or more members. A simple rotation schedule is enough to ensure that all satellite passes are utilized (see Figure 1.2 for an image Cal Poly's two amateur earth station setups).

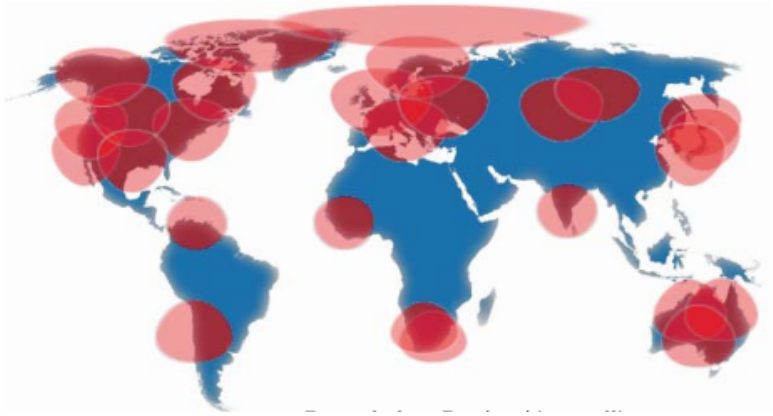


Figure 1.3: LEO Ground Coverage Using the GENSO Network [31]

While the current CubeSat operations situation does not require automated operations, the Global Educational Network for Satellite Operators (GENSO) will soon greatly increase the potential operations time [13]. GENSO is a project which promises increased educational satellite (i.e. CubeSats) communication time by connecting earth stations all over the world through the Internet [41]. For example, when Cal Poly's CP3 is within communication range of the University of Aalborg, Denmark's earth station, Cal Poly can command CP3 via the Internet through Aalborg's station. An important point is that earth station sharing is

bidirectional so that when Aalborg's satellite AAUSat-II is within communication range of Cal Poly, Aalborg can command their satellite using Cal Poly's earth station. See Figure 1.3 for example coverage with 27 ground stations on the GENSO network.

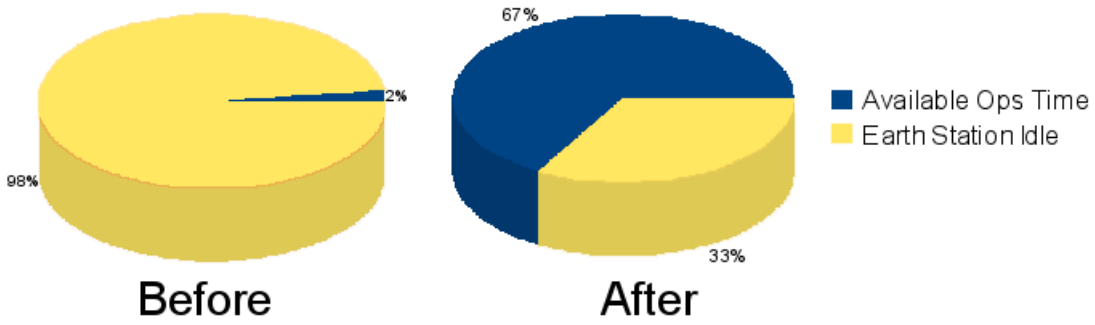


Figure 1.4: Time Available per Day to Conduct Ops Before (left) and After (right) GENSO

When GENSO is completed, it will increase satellite communication time from 30 minutes per day to approximately 16 hours per day which will proportionally raise the amount of manpower required for operations by 32 times (see Figure 1.4). Since this substantial increase cannot feasibly be compensated by adding more student labor, another solution, automation, must be explored for educational satellites.

1.1 Thesis Outline

Before the specifics of this thesis' research is discussed, an outline is provided. To define the scope of this thesis and to determine what work still needed to be completed in the field of autonomous ground operations, a thorough background search was conducted. The prior autonomous operations systems were then vali-

dated using a validation framework developed as part of this thesis based on five system attributes. The prior systems were placed into the validation framework to more accurately determine which aspects were lacking. Once this validation is complete, this thesis describes the software framework created to quickly develop additional autonomous ground operation systems. Using this software framework, three autonomous systems were developed for both Cal Poly's CP6 CubeSat and the University of Tokyo's XI-IV CubeSat. The autonomous systems were then verified and validated using the previously described validation framework. The thesis then concludes with the potential future work based on the contributions of this thesis.

Figure 1.5: Image Taken by AeroCube-2 of Cal Poly's CP4 in Space

1.2 Use of General Terms

The terms below are defined for use in this thesis as they have multiple meanings in today's English vernacular. A glossary of additional terms can be found in Appendix A.

- **Lights-Out Operations**: Operation of a ground control center without the presence or direct intervention of people [47].
- **Satellite**: An object launched to orbit Earth or another celestial body [6]. At the time of this writing, this includes all CubeSats. The more general term spacecraft is defined below.
- **Spacecraft**: Throughout this thesis, the word satellite will be used although all instances can be replaced more generally with the term spacecraft (a vehicle designed for travel or operation in space [7]). This is possible since the developed Autonomous Satellite Operations Framework (ASOF) makes no distinction.

Chapter 2

Literature Review

The following section first introduces a validation framework which can be used to judge all autonomous operations systems. Previously researched autonomous operations systems are then reviewed and validated against the aforementioned validation framework. Introducing these prior systems will define what has already been accomplished at a professional level and outline possibilities for autonomous CubeSat operations (see Figure 2.1 for an overview of the prior systems).

2.1 Evaluation of Existing Systems

Before the existing systems can be reviewed, a validation framework must be established in order to objectively compare each system. The framework must consist of metrics that summarize the systems' requirements allowing members of the field to clearly see what work has yet to be done correctly. Brann created such a framework for the evaluation of the GENIE system in 1996 [1]. Brann's

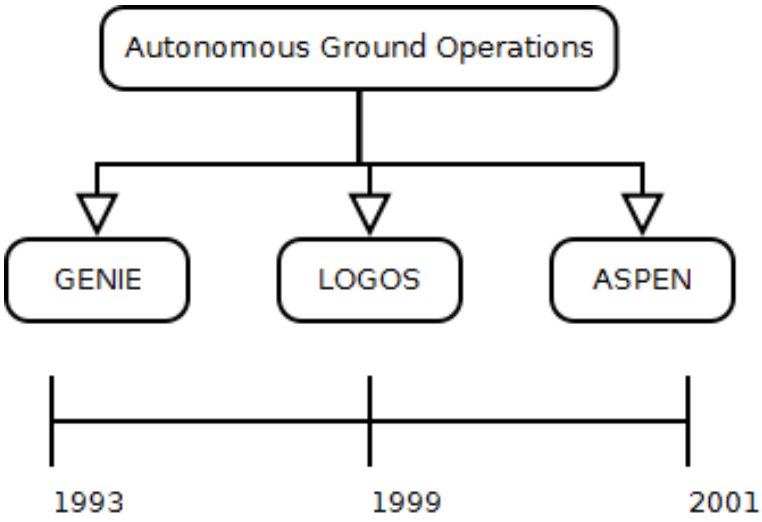


Figure 2.1: The Previous Work Being Reviewed

paper outlines the following criteria for an autonomous system (a sixth criteria, timing constraints, is excluded due to its inapplicability to the analyzed systems).

1. **Inspectable**: What is the system currently doing and why is it doing it?
2. **Predictable**: What will the system do next?
3. **Repairable**: Can the system be restarted with minimal effort?
4. **Extensible**: Can the system be easily improved?
5. **Intelligent**: Does the system learn from its mistakes?

While these attributes are able to differentiate autonomous operations systems, a simple binary scoring does not suffice. In order to add a quantitative aspect to the validation framework's scoring system, the following four scores will be used.

N/I = Not Implemented, P = Poor, S = Satisfactory, G = Good

In order for the validation framework to remain objective, each scoring option must be explicitly defined for each attribute in Brann's criteria. These scoring definitions are listed below for each attribute.

2.1.1 Inspectable Scoring Definition

The different inspectability scores quantify how much of the autonomous system's internals are displayed to the user.

- **Not Implemented:** No display is provided to the user.
- **Poor:** Only the system's inputs and outputs are displayed.
- **Satisfactory:** High level operations tasks are shown but small details such as individual uplinks are not displayed.
- **Good:** Decisions made by the system are displayed with reasonable justification.

2.1.2 Predictable Scoring Definition

The predictability scores quantifies how foreseeable the system's actions are to a human operator.

- **Not Implemented:** Actions appear to be randomly generated.
- **Poor:** The system generally follows a predictable path with some deviations during nominal operations.
- **Satisfactory:** Decisions are predictable except for decisions made in response to errors conditions.

- **Good:** Decisions are predictable including decisions made in errors conditions.

2.1.3 Repairable Scoring Definition

Repairability scores quantify how easy it is to fix and restart an autonomous operations system when an unrecoverable error occurs.

- **Not Implemented:** When a fatal error occurs, the system must be completely restarted from the beginning of the task sequence.
- **Poor:** When a fatal error occurs, the system can be restarted at some point during the last executing task with some lost progress.
- **Satisfactory:** When a fatal error occurs, the system can be restarted with no lost progress but the repair requires programmer involvement.
- **Good:** When a fatal error occurs, the system can be restarted with no lost progress and the repair can be made by a mission operator.

2.1.4 Extensible Scoring Definition

The extensibility score quantifies how easy it is for changes to be introduced to the autonomous operations system.

- **Not Implemented:** A compiled binary is distributed which allows for no modifications.
- **Poor:** Modifications require editing and a recompilation of the source code.

- **Satisfactory:** Modifications can be made via a configuration file but require programmer involvement.
- **Good:** Modifications can be made via a configuration file and can be completed by a mission operator.

2.1.5 Intelligent Scoring Definition

The intelligence scores quantify to what degree an autonomous operations system is able to modify its own behavior and remember these new behaviors.

- **Not Implemented:** No attempt to modify behavior based on prior executions. A human is required to make all behavior modifications.
- **Poor:** A flexible programming model is used that changes its behavior based on system inputs. No behavior modifications are saved to the system for future executions.
- **Satisfactory:** Behavior modifications are dynamically generated and stored in the system across executions. Modified behaviors are generated and stored offline using log information from prior executions.
- **Good:** Behavior modifications are dynamically generated and stored in the system across executions. Modified behaviors are generated and stored online such that behaviors learned during execution can be used later in that same execution.

Now that the validation framework has been explicitly defined for each of Brann's attributes, three prior autonomous operations systems will be introduced and evaluated using this framework.

2.2 GENIE

This section on the Generic Inferential Executor (GENIE) is generally cited from Hartley [17]. GENIE is NASA's first attempt to automate spacecraft operations. In 1993, GENIE started at NASA Goddard Space Flight Center (GSFC) with the goal of replacing the spacecraft command operator by programming his/her actions into GENIE. If GENIE was successful, it would have replaced at least one pass operator, therefore reducing the required manpower for operations [17].

NASA command centers at the time were typically staffed with two people per satellite per shift [17]. The first person acts as a Command Controller while the other is the Spacecraft Analyst [17]. The Command Controller is responsible for selecting which commands to send and the Spacecraft Analyst monitors the spacecraft's health. The Spacecraft Analyst then has the ability to stop the transmission of commands if a problem is discovered. In this setup, GENIE completely replaces the Command Controller role reducing the required man power for operations by 50%.

For testing and to allow for multiple uses, GENIE supports three different operating modes [17]. The mode with the least control is called *Shadow Mode* which displays the next command GENIE would send if it had control of the earth station. The next mode is *Advisory Mode* which displays the next command to send and sends that command once a human operator approves. An operator approves the command by clicking the send button on GENIE's GUI. *Advisory Mode* is close to the fully automated system, but allows a human to verify the commands that GENIE sends. The final operating mode is *Controlled Automation* which gives full spacecraft control to GENIE. It is the *Controlled*

Automation mode which enables GENIE to conduct Lights-Out Operations.

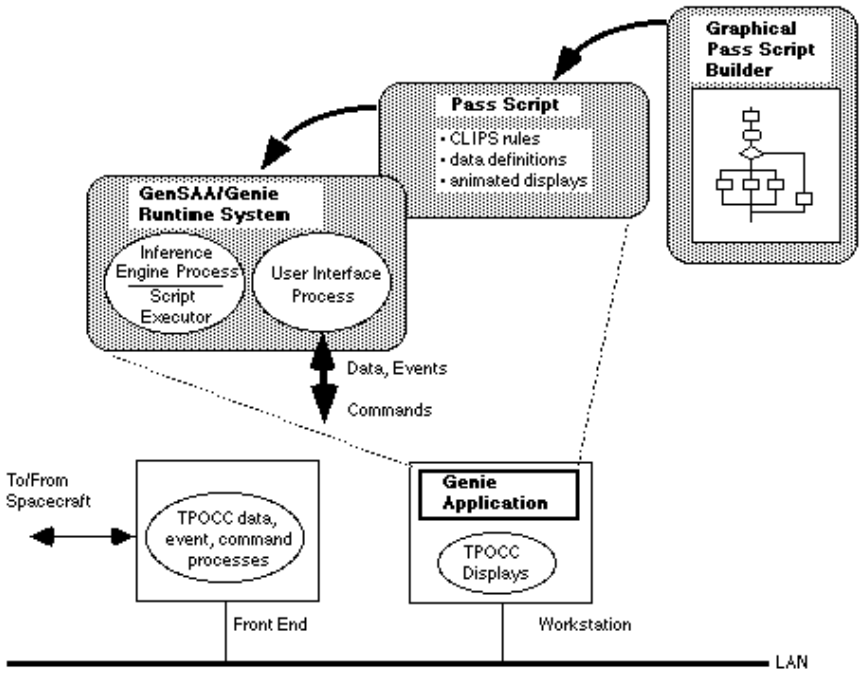


Figure 2.2: The Three Components (Top) of the GENIE Application

The GENIE application is split into three distinct parts (see Figure 2.2). The monitoring portion of GENIE is a modified instance of NASA Goddard's Generic Spacecraft Analyst Assistant (GenSAA) [21]. GenSAA is a program which monitors incoming telemetry and uses user-defined rules to check for abnormalities onboard the spacecraft. These rules are created with a graphical interface which is easily used by spacecraft engineers rather than computer programmers, minimizing errors due to human miscommunication. GenSAA uses the C Language Integrated Production System (CLIPS) [39] to define the monitoring rules. When GenSAA detects a potential problem onboard the spacecraft, it pages a member of the operations team so that he/she can investigate the problem. This notification allows potentially fatal problems such as low voltage errors to be corrected before any damage can become permanent.

The second component of the GENIE system is the graphical Pass Builder. The Pass Builder allows an operations team member to define what actions need to be completed during the pass. The Pass Builder creates a flow chart of all spacecraft activities for the pass and their following activity based on the spacecraft's response.

The third component is the Pass Executor. This component takes a pass script built with the Pass Builder and uses it to command the spacecraft. The Pass Executor itself is built within the rule-based system CLIPS. This component also uses a graphical display to show GENIE's current location in the pass script so that observing operations team members can verify GENIE's correctness.

2.2.1 Advantages

The following is a list of the GENIE system's advantages.

1. **Notification of Problems**: GENIE automatically notifies flight operation team members with a page when a problem is discovered onboard the spacecraft. This has the potential to prevent fatal spacecraft problems (e.g. battery failure).

2. **Three Modes of Operation**: GENIE is able to be used in different situations due to its three operating modes with different levels of automation. These modes make the transition to Lights-Out Operations easier since trust in the GENIE system can be incrementally developed.

3. **Graphical Script Creation**: GENIE's ability to use graphically created pass scripts enables spacecraft engineers, rather than programmers, to create the pass scripts which GENIE executes. Removing the programmer

from the process reduces spacecraft activity encoding errors during pass script creation. This reduction in encoding errors results since the satellite operator no longer needs to communicate through the programmer.

4. **Graphical Execution**: To allow for transparency of the GENIE system, GENIE provides an animated graphic of pass script execution during run-time. Present flight operations team members are able to verify GENIE's behavior to increase confidence in the system.

5. **Action Log**: GENIE outputs a log of all of its actions so that if a problem occurs with the spacecraft, GENIE's actions can be reviewed for potential flaws.

2.2.2 Disadvantages

The following is a list of the GENIE system's disadvantages.

1. **No Relation Between Passes**: Currently pass scripts are completely independent from one another. That is if all of the pass script's activities are not able to be completed within a given pass, the entire pass script must be executed again during the next spacecraft pass. This is inefficient and could lead to duplicate commands being sent which may harm the spacecraft.

2. **Unforgiving Timing System**: Currently, GENIE divides all activities in a pass script into time segments. In order for the pass to be completed, all activities must be finished within their time segment. If the activity cannot be completed within the allotted time the pass fails. This behavior is not a correct imitation of a satellite operator since if an activity takes 31 seconds

instead of 30 seconds to execute, a human operator would not consider this a pass failure [1].

3. **No Pass Script Modularity**: Pass scripts are created as one large file storing all spacecraft activities. GENIE does not support building subroutines in different files which could then be included into other pass scripts. Adding this functionality would allow for more modular testing of pass scripts and reduce the script creation time through reuse of subroutines.

4. **Fails Gracelessly**: When a problem occurs in GENIE, the GENIE application stops operating and waits for a human to manually restart the system. This behavior is not supportive of Light-Out Operations which cannot rely on human intervention when problems occur.

5. **No Learning Capabilities**: GENIE is unable to automatically learn from its mistakes. For instance if GENIE sends a particular command which makes the spacecraft become unresponsive for a period of time, the only way to stop GENIE from continuing this behavior is to have a spacecraft engineer modify the pass script accordingly.

2.2.3 Validation Framework Results

The following is an evaluation of the GENIE system using the established validation framework.

- **Inspectable: Satisfactory**

  The GENIE system only shows its users the Pass Script that was created by the Pass Builder. This visualization displays all of the information regarding the high level tasks that the system is completing.

- **Predictable: Satisfactory**

  The GENIE system follows a logical progression while it completes its tasks. There is, however, limited autonomous error recovery making its behavior during error situations unpredictable.

- **Repairable: Not Implemented**

  GENIE does not provide much information to fix errors that occur and does not allow a human operator to intervene. After the error is thought to be corrected, the GENIE system then has to be restarted from the beginning of the pass script.

- **Extensible: Good**

  The GENIE system allows changes to its pass scripts via the Pass Builder. Additionally, CLIPS rules can be added to GenSaa to extends its functionality. Since both of these procedures are graphical and straightforward, a mission operator can make these changes.

- **Intelligent: Not Implemented**

  GENIE does not allow for any behavior modifications to exist across system executions.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
GENIE	Satisfactory	Satisfactory	Not Implemented	Good	Not Implemented	Rules

Table 2.1: GENIE's Evaluation Using the Validation Framework

2.3 LOGOS

This section on the Lights-Out Ground Operations System (LOGOS) is generally cited from Truszkowski [47]. NASA Goddard's next attempt to implement automated spacecraft operations was LOGOS. LOGOS is a large scale automation system which not only includes automated spacecraft command using GENIE, but also automates all other ground station functionality (see Figure 2.3 for LOGOS' general concept diagram).

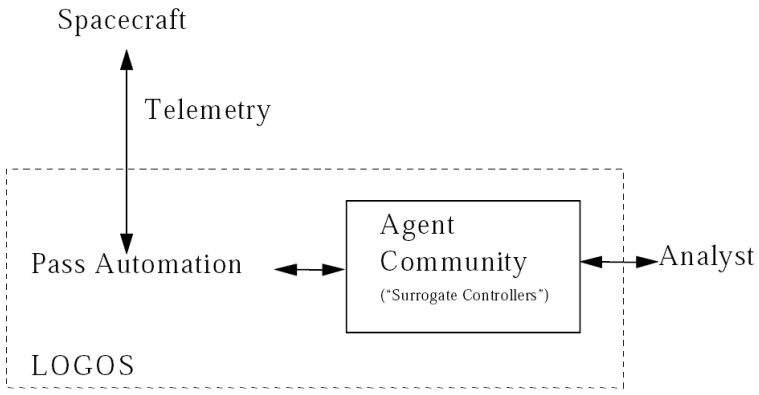


Figure 2.3: The LOGOS Concept

LOGOS is developed as a collection of software programs, each with a particular responsibility. Some software programs, also called agents, are responsible for spacecraft health monitoring while others are responsible for error analysis and recovery. Agents in the system are considered "tool users" which allows them to utilize existing tools to mimic human behavior more closely. For example, the GenSAA/GENIE agent (LOGOS embeds and uses the previously introduced GENIE system) uses the GenSAA and GENIE system directly as if the agent was a flight operations team member. This architecture allows for a plug-in agent framework which makes adding agents easy (see Figure 2.4 for a diagram of LOGOS' framework).

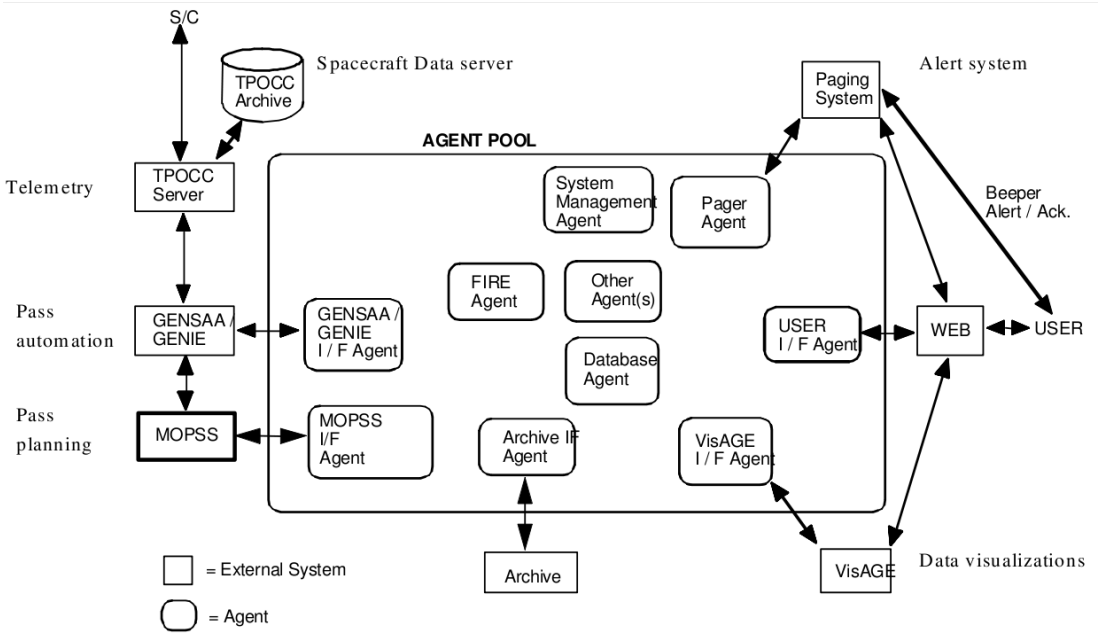


Figure 2.4: The LOGOS Framework

Agents in LOGOS are able to communicate with each other using the Agent Communication Language (ACL). For instance, the GenSAA/GENIE agent is able to notify the error diagnosis and resolution agent when problems occur using the ACL. The ACL is also extended to humans when an unrecoverable error is found. Specifically, the UserIFAgent pages to a human operator who can then log into the system and work among the agents to resolve the problem. Ideally all agents would be able to learn from this human interaction but the LOGOS system does not currently support this functionality.

2.3.1 Advantages

The following is a list of the LOGOS system's advantages.

1. **Agent Modularity**: Since all earth station operation responsibility is com-

partmentalized across many different agents, maintenance is easy as agents are designed to have high cohesion. The agent framework also enables new agents to be quickly added to the system.

2. **Problem Resolution Facility**: Unlike GENIE, LOGOS implements a secondary problem resolution agent which examines any errors reported from GenSAA/GENIE and tries to solve the problem itself before contacting external help. This extra step of problem resolution allows for small problems to be resolved automatically, increasing the system's robustness and reducing the need for human intervention.

3. **Agent Communication**: Instead of each agent being independent, agents are allowed to communicate with one another using the ACL. This design allows for interesting cooperative behavior between agents that evolves during the use of the system. These cooperative relationships may eventually reveal patterns or trends in spacecraft operations that were not previously recognized.

2.3.2 Disadvantages

The following is a list of the LOGOS system's disadvantages.

1. **Scalability**: One issue with using LOGOS' agent framework is scalability. Currently when one agent learns information, it is able to communicate that data to other agents via the ACL. As agents are added to the system, the number of possible communication channels grow quadratically. While this might not be a concern for the current number of agents, problems may occur as the LOGOS system is extended.

2. **Increased Overhead**: Due to the system's distributed nature over many agents, there is overhead associated with internal communication. While this slight decrease in efficiency is a disadvantage, the alternative is to combine all agents into one large agent which decreases code modularity. Therefore even though this overhead is a disadvantage, it may be tolerated for increased modularity and maintainability.

3. **No Learning Capability**: While the creators of LOGOS plan to include agent learning through human interaction with the system, this functionality is not yet available. Giving all agents the ability to learn enables the system to improve over time without direct human maintenance.

2.3.3 Validation Framework Results

The following is an evaluation of the LOGOS system using the established validation framework.

- **Inspectable: Satisfactory**

  The LOGOS system displays the high level tasks to be accomplished. LOGOS does not, however, display the complex interactions between its agents.

- **Predictable: Good**

  Since LOGOS uses GENIE for its execution, LOGOS has good predictability during nominal conditions. With the addition of an error recovery agent, the system is even predictable during error situations.

- **Repairable: Satisfactory**

  When the LOGOS system fails, a human operator is notified of the problem.

To resolve the problem, however, a knowledge engineer must log into the system as a LOGOS agent and control the other agents using the ACL.

- **Extensible: Poor**

  For the LOGOS system to be extended, a programmer must either create or modify an agent in the agent pool. This requires the writing/modifying of a LOGOS agent's source code.

- **Intelligent: Not Implemented**

  While agent learning in LOGOS was discussed as a needed improvement, LOGOS does not have the ability to remember information across executions.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
LOGOS	Satisfactory	Good	Satisfactory	Poor	Not Implemented	Rules

Table 2.2: LOGOS' Evaluation Using the Validation Framework

2.4 ASPEN

This section on the Automated Scheduling and Planning Environment (ASPEN) is generally cited from Chien [2]. Since all the previous systems use the GENIE application, they all contain the same base problems such as operating system dependency and unforgiving time constraints. In order to correct these problems, Chien at NASA's JPL has created ASPEN [2]. ASPEN's job as a planner/scheduler is to "accept high-level goals and generate a set of low-level activities that satisfies the goals, do not violate any of the spacecraft's flight rules or constraints, and optimize the quality of the plan" [2]. ASPEN does this

by decomposing a given goal into different high-level tasks (see Figure 2.5 for a visualization of ASPEN's goal decomposition).

Figure 2.5: ASPEN's GUI Showing Goal Decomposition

While ASPEN could construct a complete and detailed plan, it instead leaves high-level tasks abstracted until a time closer to execution. This level of abstraction allows ASPEN to plan but not commit to a particular method of completing the task. For instance if a spacecraft is required to point towards the earth, the spacecraft can either be commanded to use its reaction wheels [30] or magnetorquers [51]. Had ASPEN selected to use reaction wheels at the time of plan generation and prior to execution, the reaction wheels had become momentum

saturated, a resource conflict would exist in the plan. Instead by postponing the task's specifics, ASPEN waits until execution and observes that the reaction wheels are unavailable and decides to instead use the magnetorquers.

If a plan is created that does result in a conflict, however, ASPEN contains a real-time planning component called the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) [2]. CASPER allows a plan generated by ASPEN to be modified during its execution, if necessary, due to unforeseen conflicts. Had ASPEN selected the reaction wheels in the previous example, CASPER would be able to detect the conflict and resolve it by changing the plan to alternatively use the magnetorquers. CASPER allows for robust plan execution that recovers from conflicts, conflicts which would have halted the previous systems and required human intervention.

In addition to automating spacecraft operations on earth, ASPEN is created generically so that it can be used on a variety of platforms. ASPEN has been used directly onboard many spacecraft to automate how commands are executed. For instance, ASPEN has been utilized on the Earth Orbiter 1 (EO-1) [40] to allow the spacecraft to change its current operating task. For instance if EO-1 is monitoring a low priority atmospheric event and it senses a volcanic eruption, EO-1 will postpone monitoring the atmospheric event and record data associated with the eruption. ASPEN's ability to modify its currently scheduled plan enables EO-1 to maximize its collected science data. In addition to spacecraft, ASPEN has also be adapted for planetary rovers [10] and unmanned aerial vehicles [33].

2.4.1 Advantages

The following is a list of the ASPEN system's advantages.

1. **Iterative Repair**: Since ASPEN commits to a high-level task plan early and allows for changes during execution, a current plan is always available. This ability enables ASPEN's high tolerance to unexpected complications. Where as the previous systems would consider a pass a failure if its current plan was unable to be executed, CASPER allows APSEN to modify the plan and continue.

2. **Extensibility**: One of ASPEN's best features is its ability to be used not just with an earth station but also onboard a spacecraft. Using ASPEN on both the earth station and the spacecraft may allow for increased robustness as both systems would be capable of adjusting the current plan at varying levels of abstraction.

3. **Plan Optimization**: When determining the best initial plan for a given goal, ASPEN uses "improvement experts" to optimize the plan. An improvement expert can be thought of as an utility function that evaluates a given plan for a particular variable. For instance, there is an improvement expert for time dependency which minimizes a plan's potential for timing conflicts. A collection of improvement experts equates to a minimization problem over a number of variables to optimize for a plan. This is an effective method of plan optimization as it evaluates possible plans based on many different considerations.

2.4.2 Disadvantages

The following is a list of the ASPEN system's disadvantages.

1. **No Learning Capability**: While ASPEN is a great improvement com-

pared to the previously discussed systems, it still does not implement any type of learning capabilities. Learning would enable ASPEN to generate better plans by remembering conflicts discovered in previous ones.

2. **Local Conflict Resolution**: When ASPEN detects a conflict in its current schedule, it searches locally within the current high-level task to resolve the problem [40]. Searching locally, however, limits the possible solutions as it does not consider reordering high-level tasks. For example, if a given high-level task requires a large amount of power but the spacecraft does not have enough at that moment, the task would fail. Had the next task been low power and have no direct dependency on the high power task, a reordering may have given the spacecraft enough time to charge its batteries to complete both tasks successfully.

3. **Limited Modeling Language**: While the creators of ASPEN believe that their modeling language is expressive, the engineers adapting ASPEN to execute on EO-1 had problems modeling the complex interaction between spacecraft subsystems. Specifically, they were unable to correctly model the interaction between EO-1's solar array and the batteries [40]. An augmented modeling language which allows for more complex spacecraft modeling would improve the system's effectiveness when prioritizing activities.

2.4.3 Validation Framework Results

The following is an evaluation of the ASPEN system using the established validation framework.

- **Inspectable: Good**

  The ASPEN system shows the user all aspects of the current plan along with the information required to make that plan.

- **Predictable: Good**

  In the nominal case, ASPEN logically follows a planning algorithm to efficiently schedule tasks. When errors occur, the tasks are replanned accordingly in a logical fashion.

- **Repairable: Satisfactory**

  The APSEN system is able to restart without any lost progress since the planning component breaks goals into small individual subtasks which can be individually scheduled. Programmers, however, are required to make changes to correct the fatal error.

- **Extensible: Poor**

  In order for ASPEN to be extended, system modifications at a source level are required.

- **Intelligent: Poor**

  ASPEN is able to dynamically reorder tasks and change its behavior to react to current conditions. This ability gives ASPEN some intelligence but ASPEN is incapable of remembering decisions made across executions to improve its performance.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
ASPEN	Good	Good	Satisfactory	Poor	Poor	Rules

Table 2.3: ASPEN's Evaluation Using the Validation Framework

2.5 Summary of Existing Research

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
GENIE	Satisfactory	Satisfactory	Not Implemented	Good	Not Implemented	Rules
LOGOS	Satisfactory	Good	Satisfactory	Poor	Not Implemented	Rules
ASPEN	Good	Good	Satisfactory	Poor	Poor	Rules

Table 2.4: Validation Framework Summary for Prior Systems

Chapter 3

Automation Framework

In order to quickly develop and evaluate autonomous operation systems, a software framework named the Autonomous Satellite Operations Framework (ASOF) was created. This framework allows AI developers to focus on creating the autonomous component of the system while ignoring standard operation configurations such as hardware drivers. To enable flexibility, the software framework is implemented in Java and consists of five modular components. These components are the Agent, the Knowledge Base (KB), the Terminal Node Controller (TNC), the Line of Sight Executive (LOSE) and the Task File. Each of these components are further described in this section (see Figure 3.1 for a UML diagram of the framework).

3.1 The Agent

For all satellite operations, specific activities are always required. These actions include data logging and satellite communication. This core set of required

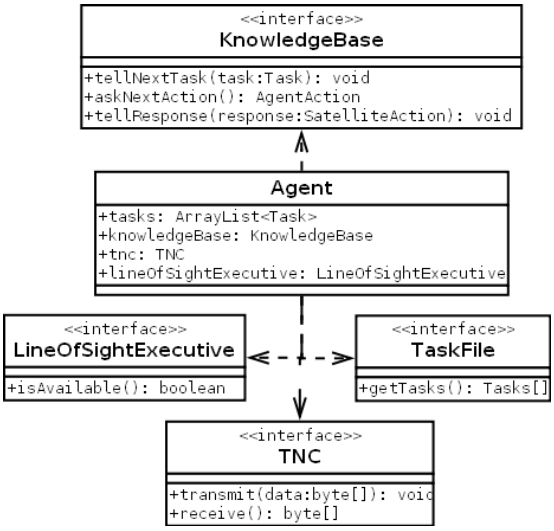


Figure 3.1: A Design Overview for the ASOF Framework

functionality is contained within in the Agent. The main functions provided by the Agent are

1. Transmit and log uplink packets
2. Receive and log downlink packets
3. Read a task list from a file

The Agent can be further extended via Java's inheritance to incorporate additional functionality.

3.2 The Knowledge Base Interface

All intelligence for satellite operations is located in the Knowledge Base. The Knowledge Base is the main framework component which when implemented, creates an autonomous system for a particular satellite. The following methods

provide the interface between the Knowledge Base and the Agent (see Section 3.6 for Knowledge Base API usage).

1. `tellNextTask(Task) : void`

   Tells the Knowledge Base which task should be completed next as listed in the task file.

2. `askNextAction(void) : AgentAction`

   Queries the Knowledge Base for the next Agent action to execute (see Section 3.8.1 for a list of all current Agent actions).

3. `tellResponse(SatelliteAction) : void`

   Tells the Knowledge Base what response was received from the satellite (see Section 3.8.2 for a list of all current satellite responses).

Currently, a Knowledge Base must be implemented for every satellite the program will track. That is, if the framework is going to track CP3 and CP6, there must be two different Knowledge Bases. The Knowledge Bases, however, are written in Java and therefore can be constructed using inheritance to reuse code between different satellites.

3.3 The Task File

Using the framework, a mission operator can define tasks to be completed using a simple text file. These tasks must be understood by the Knowledge Base for execution. A text file is currently used and structured with each line representing a single task followed by a list of task parameters. An example task file can be seen in Figure 3.2.

```
CDHDataDump()
RunADCSExperiment(1, "Magnetometer1.test")
DumpADCSExpData("Magnetometer1.test")
RunADCSExperiment(2, "ExpTemps1.test")
RunADCSExperiment("ExpTemps1.test")
```

Figure 3.2: An Example Task File

3.4 The TNC Interface

Since most satellites communicate over a radio link, operations software must be able to modulate/demodulate data sent between the satellite and the earth station. The modulating and demodulating of data is commonly done using a terminal node controller (TNC). A TNC can be a physical device such as the KPC9612+ [23] or implemented in software with a program such as MixW [11] (see Figure 3.3 and 3.4 for screenshots of the KPC9612+ and MixW respectively).

Figure 3.3: The KPC9612+ Hardware TNC

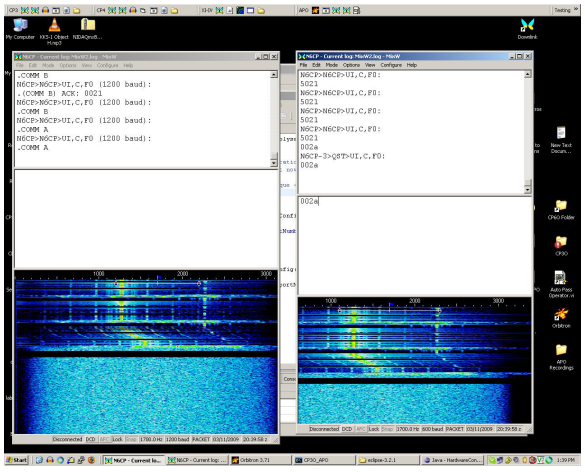


Figure 3.4: The MixW Software TNC

Since each earth station is different, a TNC driver must be implemented using the TNC interface. A default serial TNC driver is provided since most hardware

and software TNCs are implemented using serial port communication. In the future, a GENSO TNC driver will be provided to allow the framework to send and receive data from any GENSO ground station around the world.

3.5 Line of Sight Executive Interface

While external tracking software can control the Doppler shift on the satellite's radio frequency and the earth station's antenna pointing, the Agent must still know when the satellite is available for communication. The LOSE abstracts this information via an interface which can be implemented by any user defined class. A SGP4 implementation is provided by default [22]. With the release of GENSO, a GENSO LOSE will be made available to notify the framework when a satellite can communicate with any earth station on the GENSO network.

3.6 Standard Program Execution

When the Agent is started, it parses the entire input task file into a serializable in-memory data structure. The Agent then begins its basic control loop which is defined in Algorithm 1.

This algorithm executes over all tasks in the input file until they have all been completed. Once the Agent has accomplished all the tasks in the task file, it reads a similarly formatted default task file. The default task file contains a list of tasks that the Agent should do once the primary task list is complete. This concept is useful since most mission operators would prefer an active as opposed to an idle satellite. At the very least, the default task file can downlink health

Algorithm 1 The Agent's Algorithm

```
for each task t in input file do
    Tell Knowledge Base t via tellNextTask(t)
    Ask Knowledge Base next action a
    while a is not FINSHED do
        response ← Execute a
        Tell Knowledge Base response
        Ask Knowledge Base next action a
    end while
end for
```

and status data for later analysis.

3.7 Historical Data Record

In the satellite industry, satellite generated data is a valuable resource on the ground since slow communication links transmitting at approximately 1200 baud restrict how much data can be received during a pass. CP3 is able to transmit a maximum of 30KB per satellite pass [12]. Therefore, it is important that satellite data is stored in its rawest form to ensure that data never has to be retransmitted from the satellite.

One way to ensure that retransmission is never necessary is to have the framework save any received data to its log. This protects against the case where satellite data fails to parse and therefore is not saved for later analysis. With the framework log available, the parsing program can be fixed and the pass reran using the framework log as input.

3.8 Framework Actions

The following actions are used by the ASOF framework as internal representations of operation events.

3.8.1 Agent Actions

This list of Agent actions enumerates the possible actions the Agent can execute.

1. **Send_Command(commandToSend):** This action sends `commandToSend` to the satellite and does not wait for any response.

2. **Wait_For_Data(timeout):** This action waits for one data packet to be received. If no response is received in `timeout` seconds, this action returns a No_Response satellite action.

3. **Send_Command_Receive_Response(commandToSend, timeout):** This action is equivalent to a Send_Command action directly followed by a Wait_For_Data action. This action is mostly for convenience since most satellite commands return one data packet.

4. **Wait_For_Time_Period(timeToWait):** This action tells the Agent to do nothing for `timeToWait` seconds. This gives the satellite time to recover from any power issues or wait for activities which should not be interrupted.

5. **Finish():** This is the final action and tells the Agent that it has successfully completed its current task.

3.8.2 Satellite Actions

This list of satellite actions enumerates the possible actions which a satellite can take.

1. **Ack():** This action tells the earth station that their command was successfully received and executed. This Ack action is typically followed by one or more data packets.

2. **Nack():** This action is received when an error occurs while processing an earth station command.

3. **Data(responseData):** This action is used when the satellite has transmitted `responseData` down to the earth station.

4. **No_Response():** This action is received when the satellite either did not decode an Agent command or the Agent is executing a Wait_For_Data action and the satellite did not transmit any data.

Chapter 4

Implementation 1:

Rule Based System

As was shown in the literature review, all of the previous automated operations research has been built using Rule Based Systems (RBS). In order to provide a baseline for this thesis, a RBS system is built and integrated with the ASOF framework. The RBS is developed using the Java Expert System Shell (JESS) is used to manage the rule engine [27] (see Figure 4.1 for a screenshot of the RBS system).

4.1 RBS Execution

The RBS system works by executing a sequence of Agent actions on an agenda. While there is one main agenda for each task, subagendas can be inserted to accomplish a task requirement before continuing on with the main agenda. This concept is implemented using an agenda stack such that while executing

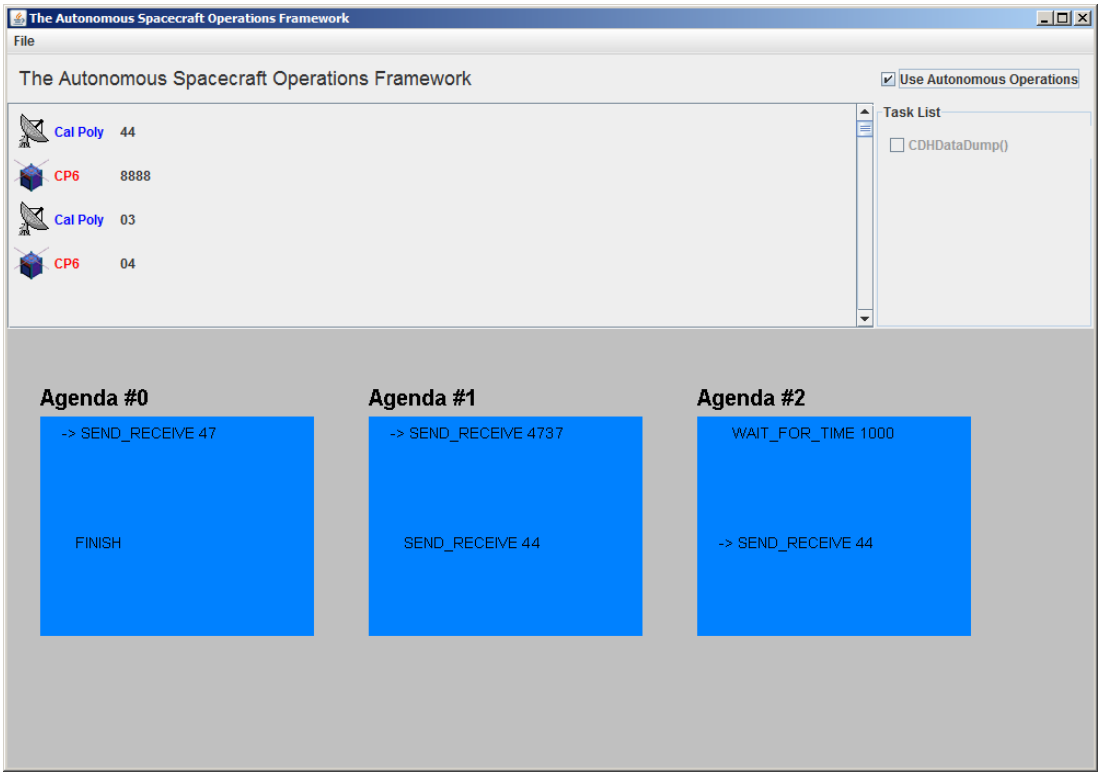


Figure 4.1: RBS Implementation Screenshot

a subagenda, another subagenda can be pushed on top of the stack. The RBS also contains three types of JESS rules. The first are the task-to-agenda rules which take a task from the task list and converts it to an agenda to complete. That is, if the Knowledge Base completes the agenda, it has completed the task. The second type of JESS rules are the preventative rules which are used to verify that a particular Agent action can be performed. These rules work to prevent harming the satellite by considering the satellite's current state. The last type of JESS rules are error recovery rules. These rules recognize and correct any onboard satellite problems given a satellite response. With the introduction of (sub)agendas and the three types of JESS rules, the RBS Knowledge Base works as follows.

Each time that the Agent calls `tellNextTask`, the RBS creates the initial agenda. The initial agenda contains the sequence of Agent action which when executed will finish the previously told task. The Agent then proceeds to ask the RBS the next action to perform via the `askNextAction` method. At this time, the JESS engine is run which enables the preventative rules to be triggered. Since JESS has access to the agenda stack, it is able to examine the top agenda's next action. The JESS rules then compare the next action with the satellite's state to verify that the action will be successful. If the JESS engine determines that there is a reason the action will not work with the current satellite state, subagendas are pushed onto the agenda stack to fix the satellite state. The RBS then returns the next action on the agenda stack to the Agent for execution.

After the Agent completes the next action, it tells the response to the RBS Knowledge Base via the `tellResponse` method. At this time the JESS engine is executed which can result in a number of outcomes. If the response is the expected response, the current action pointer on the top agenda is advanced. If this advance results in the top agenda being completed, it is popped off the agenda stack. If an error occurs, however, the error recovery rules are triggered and a corrective subagenda is pushed onto the agenda stack. The current action pointer is not advanced so that when the corrective subagenda is completed, the error causing action is executed again with the appropriate satellite state. When the original task agenda has been completed, the RBS returns the `FINISHED` Agent action which signals that the RBS is ready for the next task. This RBS execution process is further explained using the example below.

4.1.1 RBS Execution Example

Suppose the RBS Knowledge Base was told that it needed to execute the `TakePicture` task. The RBS would first create the initial task agenda found in Figure 4.2. Since the CP6 satellite does not posses complex command sequencing, the `TakePicture` task translates into one real `TakePicture` command. When the Agent calls the `askNextAction` method, the JESS engine inside the RBS is then executed. For the purposes of this example, assume that the satellite is currently not in normal ops and the payload is off. Since the `TakePicture` command requires that the satellite be in normal ops and the payload on, the JESS engine's prevention rules will recognize this problem and push two preventative subagendas onto the agenda stack (see Figure 4.3). The RBS then proceeds to execute the top agenda's next action until all agendas are complete.

Imagine again for the purposes of this example that the satellite is also in a low power situation. Since taking a picture requires a lot of data transfer onboard the satellite, a healthy power state is required. In this situation, the next time the RBS Knowledge Base tries to execute the `TakePicture` command, the JESS engine will push a preventative subagenda onto the agenda stack to wait for the batteries to charge (see Figure 4.4). When the satellite has charged to an appropriate level, the JESS engine will allow the RBS Knowledge Base to execute the `TakePicture` command thus completing the `TakePicture` task.

Suppose, however, that when the `TakePicture` command is executed, the satellite responds with a Nack(Not in Normal Ops). In this case, the JESS engine will be executed and its error recovery rules will push a corrective subagenda onto the agenda stack to put the satellite back into normal ops (see Figure 4.5). This scenario can happen for many reasons such as the satellite being reset by

radiation. After the RBS system puts the satellite back into normal ops, the `TakePicture` command can be sent which completes the `TakePicture` task.

4.2 Implementation

This section outlines the specifics of how the RBS system was implemented.

4.2.1 Satellite Model

In order for the RBS to make good operations decisions, it must always maintain a believed model of the satellite's state. To do this, the last satellite snapshot is stored along with the time it was taken. This snapshot can then be queried to answer questions about the satellite at a given current time. For instance, if the RBS wants to know if the satellite is in normal ops, it would call

```
satModel.isInNormalOps(System.currentTimeMillis())
```

which returns true if the satellite is believed to be in normal ops at the current time and false otherwise. This model is updated every time a command is issued or a response is received. The following fields represent some of the common parameters monitored for the CPX brand of satellites that are available in the snapshot.

1. Snapshot time
2. Beacon rate
3. Time left in normal operations

4. Time till payload turns off

5. Battery voltages

6. Battery temperatures

An example situation where these parameters are useful is during payload operations. All CPX satellites have the concept of a normal operations (ops) mode which is a high power state required for payload command execution. Normal ops must be enabled by sending a `GoToNormalOps` command. Since normal ops is a high power state, there is a three day inactivity timer to deactivate normal ops for safety reasons. That is, after three days of no contact from an earth station, the satellite goes back into prior-to-operations (preops) mode. Therefore, it is necessary to query the satellite model before sending a payload command to verify that the last command received by the satellite was less than three days ago to make sure the satellite is still in normal ops.

4.2.2 Task to Agenda Rules

One of the simple functions which JESS is used for is to convert a task from the task list into an agenda containing the Agent actions required to complete that task. These task-to-agenda JESS rules are straightforward but tedious to implement. For instance when the TakePicture task is told to the RBS Knowledge Base, an initial agenda is pushed onto the agenda stack (the result shown in Figure 4.2). An example JESS rule used to translate the `CDHDataDump` task into its corresponding agenda can be seen in Figure 4.6.

4.2.3 Preventative Rules

In order to protect the satellite from executing potentially harmful actions, the RBS has implemented safety checks on the ground via prevention rules (see Figure 4.7 for an example prevention rule). That is when an agenda specifies a potentially harmful action, the RBS checks the satellite model via its rules to verify that the satellite can safely complete the action. For example when the Agent is about to execute any power intensive commands, the satellite model is checked for low batteries. If there is not enough battery power available, the RBS waits for 30 seconds and then sends the status command to check if the satellite has charged to a safe battery level. If so, the power intensive command is executed and operations proceed as normal.

4.2.4 Error Recovery Rules

While the preventative JESS rules work to avoid causing errors onboard the satellite, error recovery rules work to fix the errors which do occur. These rules analyze the satellite's responses to determine if an error has occurred. For the CPX brand of satellite, No_Responses and Nacks are considered errors. If a No_Response is received, an error recovery rule fires and does not advance the agenda pointer. In this way the next time the Agent calls the `askNextAction` method, the previous command is resent. If a Nack is received by the RBS, the error recovery rules' reactions depend on the associated Nack code. For instance if the Nack(not in normal ops) is received, the error recovery rule will push a go-to-normal-ops subagenda onto the agenda stack as well as not advance the agenda pointer. As a result, the Agent will first complete the subagenda to correct the problem and then resend the command which originally caused the

Nack. This time, however, the error is corrected and progress is made on the main agenda. The error recovery rules, in this way, compliment the preventative rules (see Figure 4.8 for an example error recovery rule).

4.3 Results

Overall the RBS system accomplishes the goal of autonomous operations. The following details its advantages and disadvantages.

4.3.1 Advantages

1. **Error Recovery:** The previously defined systems implement error recovery as a fault decision tree via a rule base. Similarly, the RBS identifies any NACKs it receives and then analyzes the satellite model to determine what could have gone wrong. Once the RBS believes it knows the most likely problem, it performs the corrective action. The RBS then resumes the original action it was executing before the problem occurred.

2. **Considers Satellite State:** What satellite commands are sent not only depends on the previous sequence of commands but also on the current satellite state. For instance, sending a command which requires a lot of power should only be sent when the satellite's batteries are fully charged. The RBS implementation considers satellite state through its in-memory satellite model. This decision making process using previous commands and the current believed satellite state is more aligned with how human operators conduct operations.

3. **Scalability:** Since similar RBS systems have been developed and used

on many large NASA missions [17], RBS system scale to more complex command structures and mission tasks. This ability to grow enables a confident investment in the RBS model for future projects.

4.3.2 Disadvantages

1. **Requires Human Involvement:** While the RBS does conduct autonomous operations and is able to handle errors it has never encountered before, its creation relies on the knowledge of existing satellite operators. Satellite operators must encode the command sequences required to accomplish each task in addition to encoding the general error prevention and recovery rules. Generating a rule set might take a satellite operator weeks and the resulting rules may be incomplete. A more automated method would eliminate human slowness and errors present during the RBS creation process.

2. **Bad Visibility into the Knowledge Base Activities:** Even though the current Jess agenda and fired rules can be displayed visually, RBS lacks clear visibility into the system. This is a problem for both developers and satellite operators. For the developers, it is difficult to debug a system when it is hard visualize why certain rules have fired. For satellite operators, it is hard to see the exact RBS reasoning and why it is executing the actions it has chosen.

3. **No Learning:** Over time, it would be beneficial if the system could become more capable. That is, if the system could use its operational experience to better perform its tasks. While learning is possible for a RBS, it is outside the scope of this simple RBS implementation.

4.3.3 Validation Framework Results

The following is an evaluation of the RBS system using the established validation framework.

- **Inspectable: Satisfactory**

  The RBS system displays all tasks and Agent actions to the user via the agenda stack. The RBS does not, however, show the rules which push new subagendas onto the agenda stack. Therefore all of the high level tasks are displayed but not all of the small details.

- **Predictable: Satisfactory**

  Since the RBS does a direct conversion of tasks into Agent actions, nominal operations are completely predictable. The RBS is not predictable when a preventative or error recovery rule is fired since those decisions are based on the non-visible satellite state.

- **Repairable: Satisfactory**

  As the RBS is developed using the ASOF framework, recovery can be resumed without the loss of progress. Fixing the fatal error, however, does require a programmer to write a new error recovery rule in JESS.

- **Extensible: Satisfactory**

  Since JESS rules are defined in a separate rule file, modifications can be made without changing the source code. A programmer/knowledge engineer is required to write the modifications in the form of JESS rules.

- **Intelligent: Not Implemented**

  Since the RBS system does not remember any information from prior executions, it possesses no intelligence.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
RBS	Satisfactory	Satisfactory	Satisfactory	Satisfactory	Not Implemented	Rules

Table 4.1: The RBS' Evaluation Using the Validation Framework

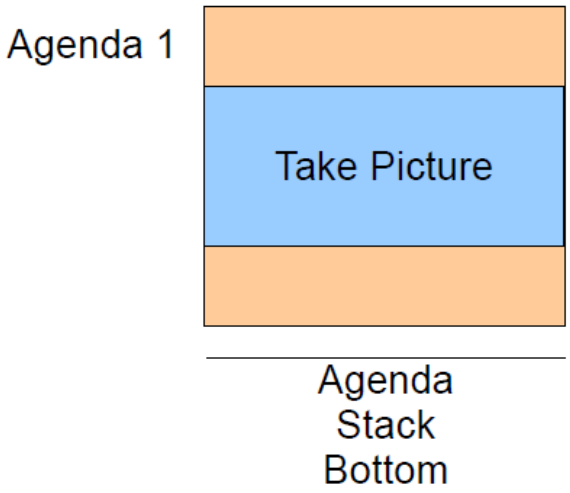


Figure 4.2: Agenda Stack Right after `tellNextTask`

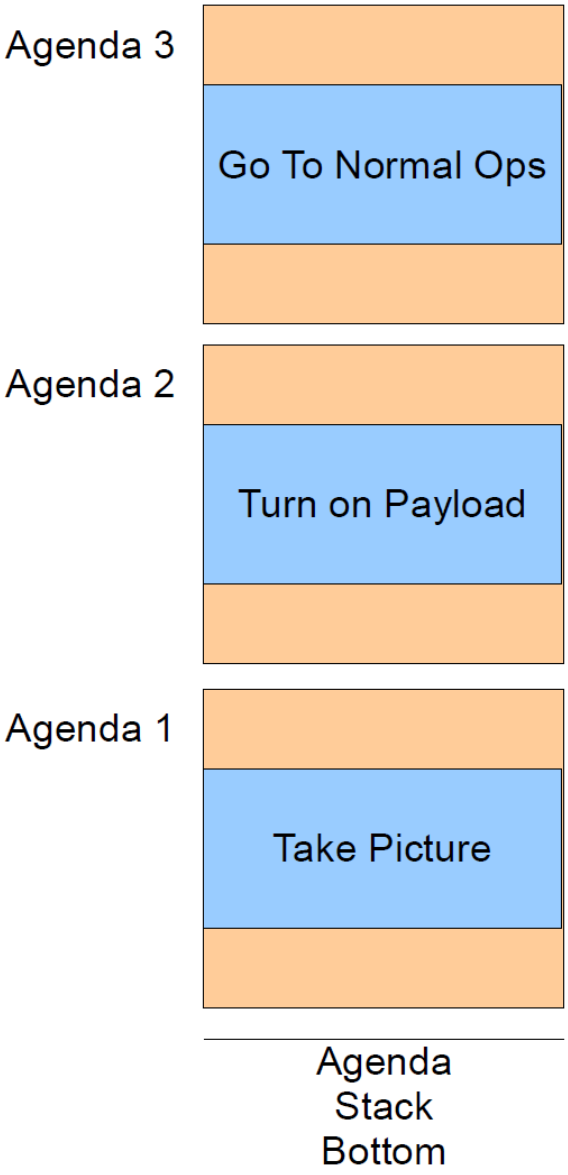


Figure 4.3: Agenda Stack while Payload is Off and Not in Normal Ops

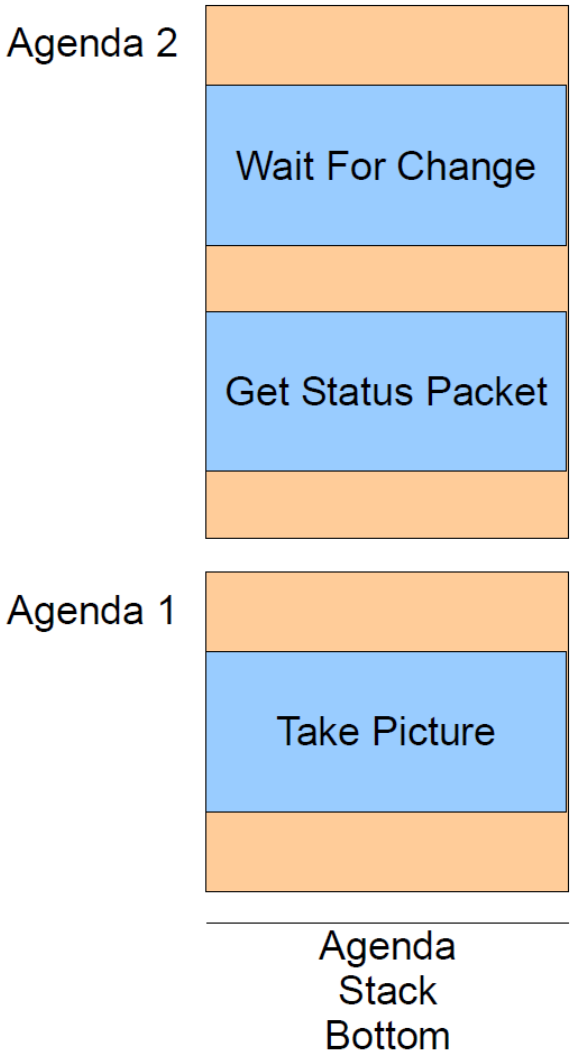


Figure 4.4: Agenda Stack with Low Power Situation

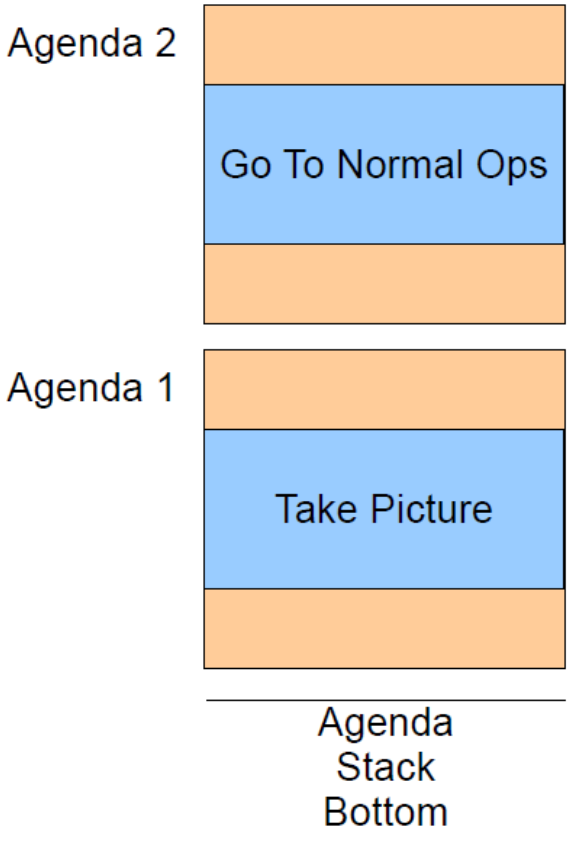


Figure 4.5: Agenda Stack after a Nack, Not in Normal Ops is Received

```
(defrule CDHDataDump

  // Get the task to complete
  (taskToComplete (taskName ?taskName))

   // Get stack
  (agendaStack (OBJECT ?agendaStackObject))

  // Make sure the task is the CDHDataDump task
  (test (call ?taskName equals ‘‘CDHDataDump’’))


=>

  // Create the new task agenda
  (bind ?taskAgenda (new Agenda))

  // Add the CDHDataDump command
  (call ?taskAgenda addSatCmd ‘‘44’’)

  // Added the finished action to the agenda
  (call ?taskAgenda addFinish)

  // Clear the agenda stack since there is a new task to execute
  (call ?agendaStackObject clear)

  // Add the new task agenda to the agenda stack
  (call ?agendaStackObject addAgenda ?taskAgenda)

  // Notify the user of the rule firing
  (printout t \"TASK-TO-AGENDA: Pushing on a Dump CDH Data \" crlf)
)
```

Figure 4.6: Task-To-Agenda JESS Rule for the CDHDataDump Task

```
(defrule normalOpsRule

  // Make sure we are in the preventative step
  (preventative)

  // Get stack
  (agendaStack (OBJECT ?agendaStackObject))

  // Make sure the action is a command
  (test (call CP6SatCmdInfo isAgentActionACommand
      (call ?agendaStackObject getNextAction)))

  // Make sure the command is a normal ops command
  (test (call CP6SatCmdInfo normalOpsCmd
      (call CP6SatCmdInfo getCommandFromAgentAction
      (call ?agendaStackObject getNextAction))))

  // Make sure we are not in normal ops
  (not (test (call CP6ReteEngine inNormalOps)))

=>

  // Add the preventative agenda
  (call ?agendaStackObject addGoToNormalOpsAgenda)

  // Notify the user of the rule firing
  (printout t \"PREVENTATIVE: Doing a Normal Ops Recovery\" crlf) "

)
```

Figure 4.7: CDHDataDump Command's Preventative Normal Ops Rule

```
(defrule nackNINO

  // Get last response
  (lastResponse (response ?satResponse))

   // Get stack
  (agendaStack (OBJECT ?agendaStackObject))

  // Check if response is a Nack(not in normal ops)
  (test (call CP6SatResponseInfo nackNINO ?satResponse))

=>

  // Add the corrective agenda
  (call ?agendaStackObject addGoToNormalOpsAgenda)

  // Notify the user of the rule firing
  (printout t \"ERROR RECOVERY: Doing a Go To Normal Ops\" crlf)

)
```

Figure 4.8: JESS Rule for Handling a Nack(Not in Normal Ops)

Chapter 5

Implementation 2:

DFA Process Model

A more straightforward method for automating satellite operations is a DFA process model which uses a Deterministic Finite Automata (DFA) [42]. Such a DFA contains Agent actions as its states and satellite actions as its transitions. This model is very expressive since it can include any of the existing Agent/satellite actions (see Section 3.8.2/3.8.2 for a list of these actions). New actions can be derived using Java's inheritance to increase the expressiveness of the DFA process model. An example DFA process model used to execute the CDHDataDump task can be seen in Figure 5.1.

5.1 DFA Process Model Execution

When the DFA process model Knowledge Base is told a task to complete via the `tellNextTask` method, the Knowledge Base locates the corresponding DFA

Figure 5.1: CDHDataDump DFA Process Model

to complete task. The Knowledge Base then sets its current state to the DFA's start state. When the `askNextAction` method is called, the Knowledge Base returns the Agent action stored in the DFA's current state (see Section 3.8.1 for a list of all available Agent actions). After the Agent executes the Agent action, the Knowledge Base is told the response via the `tellResponse` method. At this point, the current state's DFA transition which matches the response is fired. This advances the DFA's current state to another state in the DFA. If there is no transisition from the current state which matches the response, the DFA process model has failed and a human operator is required to fix the situation. This only occurs, however, when the current operations situation has never before occurred. If the situation had happened before, the interaction would have been recorded in the operations event log and extracted during the creation of the DFA process model. The DFA process model continues in this fashion until it reaches a DFA state which contains the Finished Agent action. This process is formally described in Algorithm 2.

5.2 Creation of a DFA Process Model

In order to create an autonomous system, there must be some source of operational intelligence. One of the benefits of using a DFA process model is that it can be easily created from an existing operations log. That is, the DFA process model construction procedure takes the result of a human operators' interactions with the satellite (the operations log) and uses it to reconstruct the human operators' process.

The DFA process model creation procedure is started by first preprocessing the operations log to make extraction easier. The processed log is then converted

Algorithm 2 ExecuteDFAProcessModel(TaskToExecute, DFALibrary)

```
DFA ← DFALibrary.getDFA(TaskToExecute)
curState ← DFA.getStartState()
while curState.getAgentAction() ≠ FINISHED do
    result ← Execute curState.getAgentAction()
    nextState ← null
    for each transition ∈ curState.getTransitions() do
        if result = transition then
            nextState = transition.getNextState()
            break
        end if
    end for
    if nextState = null then
        Report Error to User
    end if
    curState ← nextState
end while
```

into a MXML file which is standard for workflow extraction algorithms. The MXML file is then provided as input to the α algorithm which results in a Petri net. This Petri net representation of the process is then converted into a DFA process model through a contraction procedure. This multi-step process can be seen in Figure 5.2 and is further described below.

5.2.1 Data Source Selection

In order to dynamically create a DFA process model, a data source containing all operations events must be available. This operations event log can be stored in a database, comma separated value (CSV) file or any other data format. For CPX satellites, PolySat has a MySql [44] database named MoredBs [4] to store all operations events which occur around the world. The most important fields MoredBs records are:

1. An event identifier
2. The time the event occurred
3. Specifics of the event (ie. event parameters)

These three simple fields are enough to generate a DFA process model. For simple process model extraction techniques, the ordering of the operations events is sufficient instead of the exact event time.

5.2.2 Preprocessing

Once the data source has been selected, a number of preprocessing steps must occur to filter out data that is not useful in the extraction process. The following

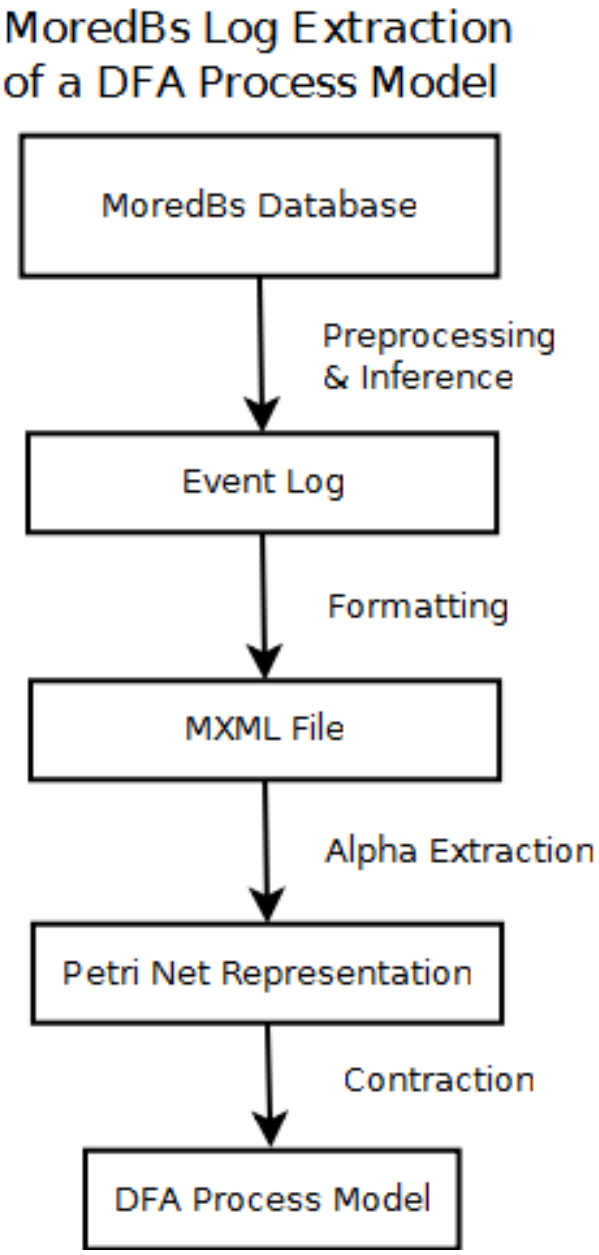


Figure 5.2: The Data Structures and Steps to Create a DFA Process Model

preprocessing steps are sufficient for correct DFA process model creation.

Beacon Removal For the purposes of task operations, beacons do not change the onboard satellite state. While beacons do tell the ground operator the state

of the satellite, beacon events are not crucial and therefore are removed to avoid confusion during process model creation.

Uplink and Downlink Passes In order for a pass to increase task operations knowledge, there must be an interaction between the ground operator and the satellite. That is, the operator must have made a conscious decision to complete a task which mean he/she sent at least one command and the satellite responded with a result. Any passes containing only uplinks or downlinks do not add to the resulting DFA process model and are filtered out during preprocessing.

Group Operation Tasks into Process Instances Since the ASOF framework uses a task list as its primary input, it is necessary to extract DFA models which pertain to one particular task. This means that the events in the log must be grouped together as task instances. Fortunately with the current CPX satellites, only one task is typically completed per satellite pass which means pass groupings are sufficient. For satellites which can accomplish many tasks per pass, it is necessary to use session detection methods [19, 18, 43] to group operational tasks into process instances.

Infer Missing Operation Events One of the most important preprocessing steps is to infer events which occurred during operations but were not logged. For instance when an operator sends a command be receives no response from the satellite, the operator will resend that command a second time. Only these two uplinks are recorded in the log and the implied `No_Response` event by the satellite is not stored. Since the operator's second uplink was a result of the `No_Response` action, it is important that the `No_Response` is represented in the

Inferred Agent Actions	Inferred Satellite Actions
WAIT_FOR_DATA	NO_RESPONSE

Table 5.1: Inferred Actions for MoredBs

log. The actions found in Table 5.1 are not recorded in the MoredBs log for the CPX satellites and have to be inferred.

5.2.3 MXML Formatting

In process extraction research, a standard file format called Mining XML (MXML) has been defined to represent event logs. MXML files are formatted in accordance with the MXML XSD [50] (see Appendix C.2 for the MXML file format). The benefit of converting the event log into a MXML file is that there exists open source implementations for many process extraction algorithms which take an MXML file as input. This prepares the event log for the next step in the DFA process model creation procedure.

5.2.4 Alpha Extraction

With an MXML version of the operations log, a process extraction algorithm can be applied to create a process model. A number of open source process extraction algorithms written in Java can be found in the Process Miner (ProM) framework which is developed by the Process Mining Group at Eindhoven Technical University [9]. The open source algorithm used for the MoredBs log is a simple process extraction method called the α-algorithm [5]. This step of the process results in a verbose, but complete Petri net representation of the operations process. More information regarding Petri nets and the α-algorithm can be found in the Appendices D and E respectively.

5.2.5 Contraction

While a Petri net representation contains all the necessary information required for executing an operations task, the Petri net is unnecessarily large (see Figure 5.3(a)). In order to reduce the amount of graph nodes, a contraction procedure is used to convert the Petri net into the final DFA process model. The contraction procedure first removes all connecting Place nodes from the Petri net. This results in a bipartite graph of Petri net transition nodes containing Agent and satellite actions. The contraction procedure then translates the Agent actions into the states of the resulting DFA process model and the satellite actions into the transitions between those states. After the contraction procedure is complete, the Petri net represented in Figure 5.3(a) becomes the DFA process model represented in Figure 5.3(b).

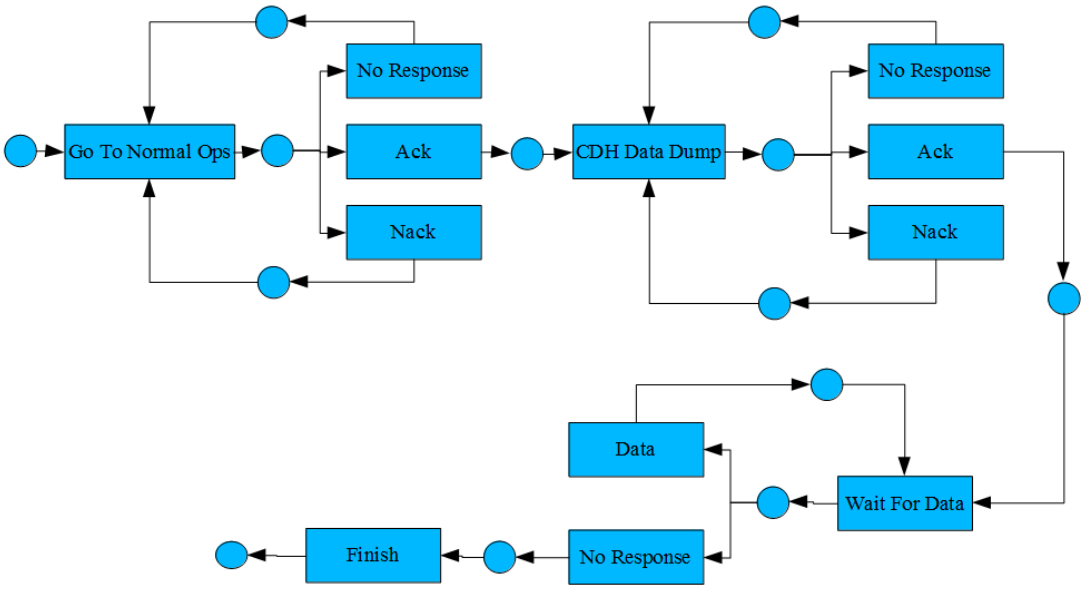
5.3 Results

Overall the DFA process model implementation accomplishes the goal of autonomous operations. The following details its advantages and disadvantages.

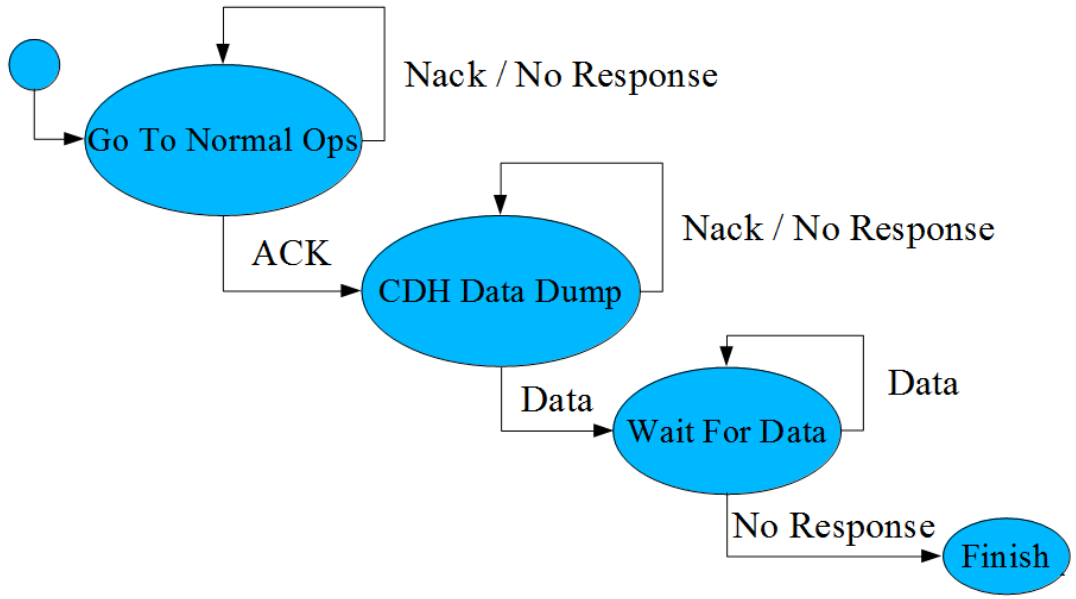
5.3.1 Advantages

1. **Automatic Creation:** Since only an existing operations log is needed to create DFA process models, no human knowledge is required. This is beneficial since most student satellite projects lack time to create an autonomous system for end-of-life operations.

2. **Easily Visualizable:** Most individuals can understand a visual represen-



(a) Petri Net of the CDHDataDump Task Before the Contraction Procedure



(b) DFA Process Model of the CDHDataDump Task After the Contraction Procedure

Figure 5.3: The CDHDataDump Task Before and After the Contraction Procedure

tation of a DFA. The states and transitions can be easily rendered using a graphics library such as Dot [25]. During DFA process model execution, color coding can be used to easily show a human observer which state the system is currently executing along with the path taken to get there (see Figure 5.4 for an example DFA process model screenshot).

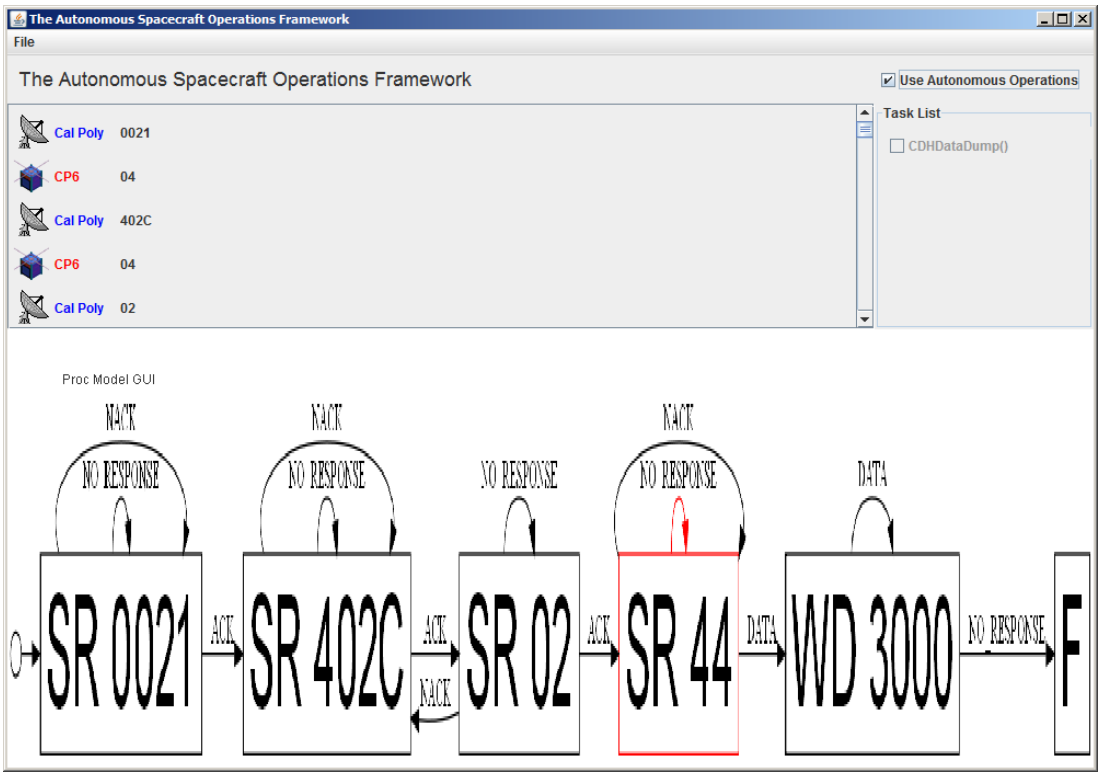


Figure 5.4: DFA Process Model Implementation Screenshot

3. **Easily Editable:** Since the DFA representation is simple, it is stored in a plain text file. This file can be easily edited by a mission operator to change any behavior which is undesirable. For instance, if a prior satellite operator performed a risky operation and it was recorded in the operations log, the extracted risky behavior can be modified or removed.

4. **Scalable:** DFA process models are bipartite graphs containing Agent and satellite actions. Since both Agent and satellite actions can be subclassed using Java's inheritance, DFA process models can be scaled to accommodate any task by deriving new actions. The bipartite nature of the graph can even be circumvented when two actions of the same type must occur sequentially (as is the case with the CDHDataDump task which uses the WAIT_FOR_DATA agent action). This functionality increases the scalability of the DFA process model Knowledge Base. This scalability is beneficial since as student satellites start to become increasingly more complex, command sequences will also become more complex.

5.3.2 Disadvantages

1. **Satellite State Not Considered:** Unlike the RBS implementation, the DFA process model does not consider the satellite's state when it is sending commands. This is problematic since there are some decisions made by satellite operators (e.g. is there enough battery power for this operation) which require the satellite state. While the extraction process can be extended to include this information, this is not currently implemented.

2. **No General Error Recovery:** Currently the DFA process model is able to recover from errors which previous operators have recovered from due to the extraction process. The DFA process model, however, is unable to recovery from never-before-seen errors. This issue is addressed in the next Knowledge Base implementation.

3. **No Learning:** Like the RBS implementation, the DFA process model has no capacity to recognize new issues, let alone fix them. The only way a DFA

process model can be modified is through a human operator modifying the DFA file which changes the DFA's behavior.

5.3.3 Validation Framework Results

The following is an evaluation of the DFA process model system using the established validation framework.

- **Inspectable: Good**

  Since all parts of the DFA process model are visible through the DFA graph representation, all activities, including error scenarios, are displayed to the user.

- **Predictable: Good**

  The DFA process model is created solely based on the prior actions of human operators so its actions are 100% predictable.

- **Repairable: Good**

  Since the DFA process model is developed using the ASOF framework, recovery can be resumed without the loss of progress. Any changes to the DFA can be easily made by editing the saved DFA file. Since the DFA file format is straightforward and easy to visualize, a mission operator can make the necessary modifications to repair the system.

- **Extensible: Good**

  The DFA process model's intelligence is stored in an easy-to-read plain text file. Any mission operator can extend the DFA process model by adding, removing, or modifying its Agent/satellite actions.

- **Intelligent: Not Implemented**

  The execution of a DFA process model statically follows the transitions of the generated DFA. At no point during its execution does the DFA process model modify its behavior based on prior executions.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
DFA Proc Model	Good	Good	Good	Good	Not Implemented	DFA

Table 5.2: The DFA Process Model's Evaluation Using the Validation Framework

Chapter 6

Implementation 3:

Hybrid Implementation

At this point, this thesis has introduced two different approaches to solving the problem of autonomous satellite operations. Each implementation has its own advantages and disadvantages which make them useful in different situations. In this chapter, a hybrid implementation is introduced which combines the techniques of the two previous implementations into a single Knowledge Base.

6.1 Implementation

Since the hybrid method is a combination of the previous two implementations, its construction is very similar. First the DFA process models must be extracted using the same process as described in Chapter 5. These DFA process models will provide the basic knowledge required for operations.

After the DFA extraction process is complete, a satellite operator is required.

The human operator does not have to specify all the commands required to complete a task since that information has already been extracted from the operations log. The human operator also does not have to write the preventative rules since they will not be used in the hybrid implementation. Instead, the operator only needs to write the error recovery rules. This dramatically reduces the required time to manually create the hybrid's RBS.

Once the DFA process models and the recovery rules have been specified, the hybrid model is ready for execution. The algorithm used by the hybrid process is exactly the same as the DFA process model algorithm (Algorithm 2) but uses the RBS when an error occurs (see Algorithm 3 for the hybrid algorithm). That is when an operation event occurs that the DFA is not able to handle, the RBS is queried for a solution (i.e. a command or action which would resolve the current unseen event). After the RBS believes that the error is resolved, the DFA resumes its activities at the point the error occurred. If the error persists, then the hybrid implementation has failed and a human operator is required.

Once the human operator understands the problem which occurred, he/she is able to modify the DFA process model or add a RBS error recovery rule which would prevent the same problem from occurring. In this way, the benefit of automatically creating the Knowledge Base for common operations is only slightly offset by the error recovery rules which need to be handwritten.

6.2 Results

The hybrid implementation requires slightly more human interaction than the pure DFA process model implementation but the added error recovery rules

Algorithm 3 The Hybrid Algorithm

```
while operating do
    Use DFA Knowledge Base
    if an error occurs then
        Use RBS Knowledge Base on last response
    end if
end while
```

make the hybrid implementation more robust than either the RBS or DFA process model by themselves. This is due to the hybrid implementation using the advantages of each implementation and diluting the disadvantages. The hybrid's specific advantages and disadvantages are listed below.

6.2.1 Advantages

1. **Easily Visualizable:** Since most of the choices made by the hybrid system are based on the DFA process model, the easily understood DFA visualization still applies. Therefore the only hard to visualize part of the hybrid system are the rules used to handle unknown errors. The error recovery rules are rarely used making the hybrid implementation overall easy to visualize.

2. **Recovery from Unexperienced Errors:** Since error recovery rules are provided via the RBS portion, the hybrid implementation is able to recover from errors which would have caused the DFA process model alone to fail. This is the primary reason for the RBS' inclusion.

3. **Only Need To Manually Make Error Recovery Rules:** The hybrid is a compromise between automated and manual creation which dramat-

ically reduces the burden placed on satellite operators. Since the tedious components of operations can be extracted from the operations log, only the error recovery rules need to be defined.

6.2.2 Disadvantages

There are still disadvantages, however, with the hybrid implementation.

1. **Satellite State Not Considered:** Similar to the DFA process model, command are sent solely based on the previous sequence of commands and satellite responses. This is the case since the RBS, which contains the error recovery rules, is never called until the DFA process model has an error. The hybrid implementation could be revised such that the RBS is called before commands are sent to incorporate satellite state.

2. **No Learning:** While a human operator can modify the DFA process model or the RBS when an error occurs, the hybrid implementation is not capable of learning from its own experiences. This is due to the fact that the hybrid implementation is a conglomeration of two non-learning implementations. The no automated learning disadvantage, however, should not greatly hinder operations since most errors should have already been seen and corrected by a human operator during critical mission operations. Their error recovery actions should be present in the operations log and extracted during the creation of the DFA process model. Additionally, most of the errors not present in the operations log will be handled by the RBS' error recovery rules. This combination leaves only a few errors which could benefit from the addition of learning.

6.2.3 Validation Framework Results

The following is an evaluation of the hybrid system using the established validation framework. Since the hybrid implementation is the same as the DFA process model implementation unless exceptional circumstances occur, their validation results closely match.

- **Inspectable: Good**

  As the hybrid implementation displays the DFA process model during execution, it too shows all decisions being made by the system.

- **Predictable: Good**

  Since the hybrid implementation follows the predicable nature of the DFA process model, it too is completely predictable.

- **Repairable: Good**

  The hybrid implementation can restart with no lost progress because it is implemented using the ASOF framework. Repairs can also be made to the system by changing the DFA files which can be accomplished by a mission operator.

- **Extensible: Good**

  The hybrid implementation can be extended in the same fashion that the DFA process model is extended, via the DFA files. As was noted in the DFA process model validation section, these modifications can be made by a mission operator.

- **Intelligent: Not Implemented**

  Since the hybrid implementation is a combination of two non-intelligent systems, it has no intelligence.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
Hybrid	Good	Good	Good	Good	Not Implemented	Both

Table 6.1: The Hybrid's Evaluation Using the Validation Framework

Chapter 7

Verification & Validation

7.1 Testing Overview

In order to test the correctness of each implementation, a set of system tests were developed. Each test is defined in a specific file structure which specifies all test parameters (see Figure 7.1 for an example test file structure). Since the framework and Knowledge Bases are developed in Java, JUnit facilitated the testing process [34] (see Figure 7.2 for an example JUnit result screen). The general test process can be seen in Algorithm 4.

Algorithm 4 The Verification Test Algorithm

Start the Satellite Simulator with the provided response file

Create an ASOF model using the provided properties files

Start the Agent using the created satellite model

Verify when the Agent has successfully completed all operations

For each implementation (RBS, DFA process model, Hybrid), three operation situations were tested.

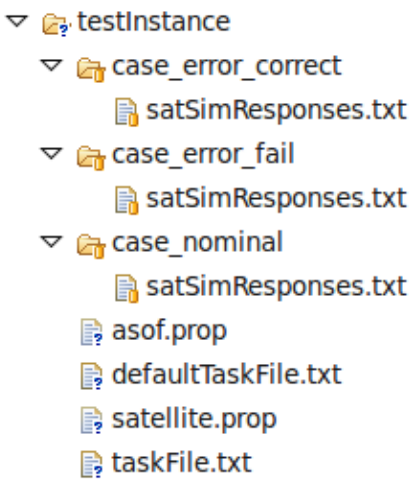


Figure 7.1: Example Test Structure

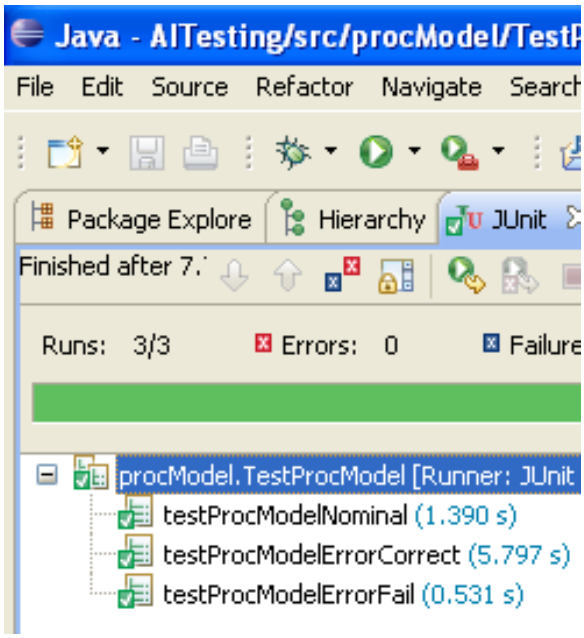


Figure 7.2: Example JUnit Verification Screen

1. **Nominal:** The nominal test was a CP6 CDHDataDump task which contained no errors during operations.

2. **Error with Recovery:** During operations, multiple Nacks and No_Responses were sent from the satellite.

3. **Error without Recovery:** An error situation in which a fatal Nack is received and the satellite is put into an unrecoverable infinite loop.

The results from these experiments can be found in Table 7.1

	Nominal	Error Correct	Error Fail
RBS	Passed	Passed	Failed
DFA Process Model	Passed	Passed	Failed
Hybrid	Passed	Passed	Failed

Table 7.1: Results of all Verification Tests with CP6

7.2 ASOF Verification With Another Satellite

Since the ASOF framework is intended to work with any satellite, a satellite besides CP6 is required for verification purposes. The University of Tokyo's XI-IV CubeSat has been in orbit since 2003 and is still operational [36]. Cal Poly has a signed agreement to operate XI-IV which makes it an ideal candidate to verify the ASOF framework. To prove ASOF's versatility, XI-IV was also executed against the previously described test cases. The results can be seen in Table 7.2.

	Nominal	**Error Correct**	**Error Fail**
RBS	Passed	Passed	Failed
DFA Process Model	Passed	Passed	Failed
Hybrid	Passed	Passed	Failed

Table 7.2: Results of all Verification Tests with IX-IV

7.3 Validation Framework Results

Now that all of the autonomous systems have been introduced and placed in the validation framework, an analysis can be completed. The following section examines the results found in Table 7.3.

	Inspectable	**Predictable**	**Repairable**	**Extensible**	**Intelligent**	**Type**
Prior Systems						
GENIE	Satisfactory	Satisfactory	Not Implemented	Good	Not Implemented	Rules
LOGOS	Satisfactory	Good	Satisfactory	Poor	Not Implemented	Rules
ASPEN	Good	Good	Satisfactory	Poor	Poor	Rules
Thesis Systems						
RBS	Satisfactory	Satisfactory	Satisfactory	Satisfactory	Not Implemented	Rules
DFA Proc Model	Good	Good	Good	Good	Not Implemented	DFA
Hybrid	Good	Good	Good	Good	Not Implemented	Both

Table 7.3: Validation Framework Summary for Autonomous Systems

Majority Rule Based As was noted in the beginning of this thesis, all of the existing systems are developed as rule based systems. Looking at the validation framework results, the non-rule based systems scored higher than the rule based systems. Therefore, it would be beneficial to investigate additional non-rule based systems for autonomous operations.

Good Inspectability and Predictability Most all of the systems have decent inspectability. This results since it is easier to debug a system which makes available its internal operations. Predictability is also high across the autonomous systems since an unpredictable system generally does not complete autonomous operations very well. Therefore to have a working system typically means to have a predictable system.

Bad Repairability and Extensibility While repairability and extensibility scored worse than inspectability and predictability, there is an explanation. Since most of these autonomous systems are prototypes, developers do not want to spend time modifying/customizing their software above the source code level. To do so would mean investing extra time in an idea which may be thrown away. Additionally, once high level configurations are written, it is harder to change the underlying system since the high level configurations lacks the expressive power of source code. Therefore, most of these prototype systems lack repairability and extensibility as defined by the validation framework.

Minimal Intelligence The ability to modify and remember behaviors is lacking in all of the reviewed autonomous systems. While the systems do accomplish the task of autonomous operations, greater intelligence would enable greater effi-

ciency and functionality. Learning and intelligence is therefore one of this thesis' and the field of autonomous operations' most important future work.

Hybrid vs DFA Process Model Using purely the validation framework, it would appear that the Hybrid and the DFA process model implementations are equivalent. This, however, is only due to a lack of resolution in the framework regarding a system's ability to recover from errors. Since the hybrid implementation primarily uses the DFA process model, the hybrid implementation can solve any problem that the DFA process model can solve. Additionally, the hybrid implementation can use the recovery rules of the RBS system when the DFA process model fails. Therefore, the added power of the RBS makes the hybrid implementation able to solve more problems than the DFA process model. This argument can be found in a more structured form in Figure 7.3. Since all other aspects are equal, the hybrid implementation is slightly more advantaged than the pure DFA process model implementation.

1. $DFAErrorsSolved$ = The amount of errors solved by the DFA process model implementation ($DFAErrorsSolved > 0$)

2. $RBSErrorsSolved$ = The amount of errors solved by the RBS' error recovery rules that are not solved by the DFA process model ($RBSErrorsSolved \geq 0$)

3. $HybridErrorsSolved = DFAErrorsSolved + RBSErrorsSolved$

4. $\therefore HybridErrorsSolved \geq DFAErrorsSolved$

Figure 7.3: Logic Showing the Hybrid Implementation has the Potential to Solve More Problems than the DFA Process Model Implementation

Chapter 8

Future Work

While a lot has been accomplished during the course of this masters thesis, there is much work that has yet to be completed. This chapter addresses the main items which still require action.

8.1 Learning Knowledge Base Library

Currently only a RBS, a DFA process model and a hybrid model are implemented with the ASOF framework. None of these implementations, however, utilize any experiences from previous executions. That is, they do not learn new or more efficient ways to accomplish tasks. In order to promote the research of learning Knowledge Bases, a set of libraries can be produced to facilitate development. These libraries would include Neural Network implementations as well as algorithms commonly found in Weka [16], Rapid Miner [38], and KNIME [35]. By lowering the barrier to entry, more people will see the benefit and develop learning Knowledge Bases for the ASOF framework.

8.2 Advanced Monitor Interface

The central concept of the ASOF framework is to greatly reduce the amount of human involvement required to operate a satellite. Continuing this thought, ASOF should be easy to monitor and make users aware when serious problems have occurred. For instance, a web monitoring interface should be created such that a human operator can open a browser and simply determine the current state of operations. Due to ASOF's Model-View-Controller architecture [26], the addition of a web view could be implemented with only minor changes to ASOF. Additionally, a human operator should receive direct notification of problems through either email or SMS so that corrective action can be taken as soon as possible.

The web monitoring interface could further increase usability by allowing a human operator to control ASOF from their browser. In this way, a human operator could receive an SMS alert on their phone and open a browser to fix any problems. These enhancements will greatly reduce the required involvement from human operators during autonomous operations.

8.3 Add HamLib Driver Support

Currently the TNC API is a generic Java interface. That is, whenever a user wants to use the ASOF framework at his/her ground station, they must write a TNC driver from scratch. Luckily this problem has already been addressed via HamLib which is a collection of drivers for the most common ground station devices (which includes TNCs) [15]. It would be advantageous for the users of ASOF to have HamLib integrated such that any HamLib driver is a selectable

TNC. Additionally since HamLib is actively being developed, when a new device becomes available, only the first individual needs to write the driver. After the driver is written once, the rest benefit from its addition to HamLib. This is the approach that the GENSO project has taken to manage ground station drivers [13].

8.4 Add Satellite State to Hybrid Implementation

Currently, the hybrid implementation does not does take any preventative measures when operating a satellite. Unlike the RBS implementation which changes its agenda based on the current satellite state, the hybrid model does no such check. This check, however, would be an easy addition and could be implemented in the following manner. Before a DFA process model Agent action is executed, the rule base is queried for any potential issues with the current action. If the RBS detects an issue, the Agent action is temporarily postponed and a corrective action is taken instead. For example if the DFA process model's next action is a power intensive CDHDataDump operation, the RBS would be queried to check if the power levels are sufficient. If not, the RBS would tell the Agent to execute the WAIT_FOR_TIME_PERIOD action. After this action is completed, the RBS would again be queried for any issues. If the power levels were at this time sufficient, the CDHDataDump operation would occur.

Chapter 9

Conclusion

This thesis addressed the problem of autonomous operations for CubeSat satellites (see Table 9.1 for a summary of this thesis' contributions). The Autonomous Framework for Satellite Operations (ASOF) was introduced as a way to rapidly develop different types of Knowledge Bases for different satellites. Using the ASOF framework, three Knowledge Base implementations were created. The Rule Based System (RBS) implementation uses the Java Expert System Shell (JESS) to conduct operations. The RBS is most similar to the existing autonomous operation systems. The second implementation creates a DFA process model using existing satellite operations logs. This dramatically reduces the time to setup autonomous operations since there is no need to have human operators write operation rules. The final implementation is a hybrid model using both the DFA process model and the error recovery rules from the RBS. The hybrid implementation receives the benefits of both while diluting the negative aspects of each, making it the best option for autonomous operations. While all these implementations successful accomplish autonomous operations, none of them learn from prior executions. Incorporating learning into these Knowledge Bases is the

next step to improve autonomous operations for CubeSat satellites.

Thesis Contributions

- Defined a quantifiable validation framework based on five evaluation criteria provided by Brann
- Surveyed existing autonomous operations systems and validated them using the validation framework
- Created the Autonomous Satellite Operations Framework (ASOF)
- Used the ASOF framework to implement three types of Knowledge Bases (RBS, DFA Process Model, and Hybrid)
- Validated the three Knowledge Base implementations with using the validation framework
- Verified the ASOF framework and three Knowledge Base implementations using Cal Poly's CP6 and University of Tokyo's IX-IV CubeSats

Table 9.1: Summary of Thesis Contributions

Bibliography

[1] D. Brann, D. Thurman, and C. Mitchell. Human Interaction with Lights-out Automation: A Field Study. In *Third Annual Symposium on Human Interaction with Complex Systems, HICS'96*. IEEE Computer Society, 1996.

[2] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, et al. Aspen–automated planning and scheduling for space mission operations. In *Space Ops*, 2000.

[3] A. Chin. CubeSat Community Website. `http://www.cubesat.org/`, March 2009.

[4] D. Cuddeback. MoredBs. `http://moredbs.atl.calpoly.edu/`, January 2010.

[5] A. de Medeiros, B. van Dongen, W. van der Aalst, and A. Weijters. Process mining: Extending the α-algorithm to mine short loops. *Eindhoven University of Technology, Eindhoven*, 2004.

[6] Dictionary.com Unabridged. Satellite. `http://dictionary.reference.com/browse/Satellite`, February 2010.

[7] Dictionary.com Unabridged. Spacecraft. `http://dictionary.reference.com/browse/Spacecraft`, February 2010.

[8] Direct TV. Direct TV: Satellite Television. http://www.directv.com/DTVAPP/index.jsp, March 2009.

[9] Eindhoven Technical University. ProM. http://prom.win.tue.nl/tools/prom, July 2009.

[10] T. Estlin, F. Fisher, D. Gaines, C. Chouinard, S. Schaffer, and I. Nesnas. Continuous planning and execution for an autonomous rover. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, 2002.

[11] N. Fedoseev. MixW - Multimode Software for Radio Amateurs. http://www.mixw.net/, November 2009.

[12] J. Foley. Personal Correspondance with Justin Foley, PolySat Project Manager, 2009.

[13] S. Forsman. GENSO. http://genso.org/, March 2009.

[14] GENIE Development Team. GENIE Introduction. http://aaaprod.gsfc.nasa.gov/gensaa/genie/, March 2009.

[15] T. H. Group. Ham Radio Control Libraries. http://hamlib.sourceforge.net/, November 2009.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[17] J. Hartley, E. Luczak, and D. Stump. Spacecraft control center automation using the Generic Inferential Executor (Genie). In *International Symposium*

on Space Mission Operations & Ground Data Systems-'SpaceOps 96', 4 th, Munich, Germany, pages 1007–1014, 1996.

[18] D. He, A. Goker, and D. Harper. Combining evidence for automatic Web session identification. *Information Processing and Management*, 38(5):727–742, 2002.

[19] X. Huang, F. Peng, A. An, and D. Schuurmans. Dynamic web log session identification with statistical language models. *Journal of the American Society for Information Science and Technology*, 55(14):1290–1303, 2004.

[20] D. Huerta. Development of a Highly Integrated Communication System for use in Low Power Space Applications. Master's thesis, California Polytechnic State University, 2006.

[21] P. M. Hughes and E. C. Luczak. The Generic Spacecraft Analyst Assistant (GenSAA): A Tool for Automating Spacecraft Monitoring with Expert Systems. *NASA Conference Publication*, 3110:129–+, 1991.

[22] D. Johnson. Personal Correspondence with David Johnson, Software Developer for Black Pepper Software, 2009.

[23] Kantronics. Kantronics KPC-9612+ Radio Modem/TNC. `http://www.kantronics.com/products/kpc9612.html`, November 2009.

[24] B. Klofas, J. Anderson, and K. Leveque. A Survey of CubeSat Communication Systems. *AMSAT Journal*, 2009.

[25] E. Koutsofios, S. North, et al. Drawing graphs with dot. Technical report, Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[26] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[27] S. N. Laboratories. JESS, the Rule Engine for the JavaTM Platform. `http://www.jessrules.com/`, November 2009.

[28] J. P. Laboratory. Jet Propulsion Laboratory: California Institute of Technology. `http://www.jpl.nasa.gov/`, March 2009.

[29] W. Larson and J. Wertz. *Space Mission Analysis and Design*. Microcosm, 1999.

[30] R. Longman, R. Lindbergt, and M. Zedd. Satellite-Mounted Robot Manipulators–New Kinematics and Reaction Moment Compensation. *The International Journal of Robotics Research*, 6(3):87, 1987.

[31] N. Melville. Global Educational Network for Satellite Operations. International Space Education Board, Hyderabad, September 2007.

[32] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[33] P. Narayan, P. Wu, D. Campbell, and R. Walker. An Intelligent Control Architecture for Unmanned Aerial Systems (UAS) in the National Airspace System (NAS). In *Australian International Aerospace Conference (AIAC), Melbourne*, 2007.

[34] Object Mentor. JUnit: Resources for Test Driven Development. `http://www.junit.org/`, February 2010.

[35] U. of Konstanz. KNIME. `http://www.knime.org/`, January 2010.

[36] U. of Tokyo. XI series totally work for 16 years. http://www.space.t.u-tokyo.ac.jp/cubesat/index-e.html, January 2010.

[37] J. Puig-Suari. PolySat. http://polysat.calpoly.edu/, November 2009.

[38] Rapid-I GmbH. Rapid Miner. http://rapid-i.com/, February 2010.

[39] G. Riley. CLIPS: A Tool for Building Expert Systems. http://clipsrules.sourceforge.net/, November 2009.

[40] R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, and A. Fukunaga. Using ASPEN to automate EO-1 activity planning. In *IEEE Aerospace Conference, 1998. Proceedings*, volume 3, 1998.

[41] G. Shirville and B. Klofas. GENSO: A Global Ground Station Network. In *Proceedings of the AMSAT-NA 21st Space Symposium*, 2007.

[42] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[43] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS Journal on Computing*, 15(2):171–190, 2003.

[44] Sun Microsystems . MySql: The world's most popular open source database. http://www.mysql.com, January 2010.

[45] R. Thompson. Correspondence with R.J. Thompson, Chief of the USGS Earth Resource Observation Systems Data Center, Sioux Falls, S.D. http://www.space.com/spacenews/archive03/landsatarch_102003.html, 2003.

[46] A. Toorian. Redesign of the Poly Picosatellite Orbital Deployer for the Dnepr Launch Vehicle. Master's thesis, California Polytechnic State University, 2007.

[47] W. Truszkowski, H. Hallock, and J. Kurien. Agent Technology from a NASA Perspective. In *Proceedings of the Third International Workshop on Cooperative Information Agents III*, pages 1–33. Springer-Verlag London, UK, 1999.

[48] W. Van der Aalst, B. Van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

[49] W. Van der Aalst, A. Weijters, and L. Maruster. Workflow mining: Which processes can be rediscovered. *Eindhoven University of Technology, Eindhoven, Beta Working Paper Series, WP*, 75, 2002.

[50] B. van Dongen. MXML: A Meta model for process mining data. `http://prom.win.tue.nl/research/wiki/_media/presentations/miningmetamodelimoa2005.ppt`, 2005.

[51] P. Wang and Y. Shtessel. Satellite Attitude Control Using Only Magnetorquers. In *American Control Conference, 1998. Proceedings of the 1998*, volume 1, 1998.

Appendix A

Glossary

This sections defines the terms used throughout this thesis.

Agent: The main processing unit of the ASOF framework which coordinates all of the other components to conduct satellite operations.

Autonomous Space Operations Framework (ASOF): The framework which enables satellite developers to quickly create Knowledge Bases in order to autonomously control their satellites.

DFA Process Model: A Deterministic Finite Automata (DFA) which encodes Agent actions as states and satellite responses as transitions for use as a model of operations.

End-Of-Life Operations: All operations which are conducted after the critical mission objectives have been completed or deemed unable to be completed.

Knowledge Base: The abstraction for all satellite specific intelligence needed for autonomous operations. When a satellite owner wants to automate their satellite operations, they start by implementing a Knowledge Base.

Lights-Out Operations: Operation of a ground control center without the presence or direct intervention of people [47].

Line of Sight Executive (LOSE): The Line of Sight Executive is queried to know if a satellite is currently available for communication. This simple interface typically uses a satellite's TLE for positioning.

MoredBs: The PolySat MySql database used to collect all packets sent to and received from an orbiting satellite [4].

Operations Event: Any event which occurs during operations. This can be anything from an operator sending a command to a response returned from a satellite.

PolySat: PolySat is Cal Poly's CubeSat program which started in 1999 [37].

Satellite: An object launched to orbit Earth or another celestial body [6]. At the time of this writing, this includes all CubeSats. The more general term spacecraft is defined below.

Satellite Pass: The interval that a spacecraft is in contact with the ground operations center [14].

Spacecraft: Throughout this thesis, the word satellite will be used although all instances can be replaced more generally with the term spacecraft (a vehicle designed for travel or operation in space [7]). This is possible since the ASOF framework makes no distinction.

Terminal Node Controller (TNC): The interface which takes binary commands from the operator and translates them to an analog signal which is transmitted over a radio channel to the satellite. The TNC also translates analog satellite responses into binary on the return path.

Two-Line Element (TLE): Two-line elements specify the numerical parameters which define a classical satellite orbit [29].

Appendix B

Satellite Simulator

In order to test the effectiveness of the ASOF framework, the result of system execution must be checked. While verification using a real satellite would be ideal, this setup is difficult and cumbersome to create. Alternatively, many issues can be discovered by simulating the satellite in software. This satellite simulator is more convenient for testing purposes since the simulator can be programmed to respond in many different ways. That is, conditions and situations can be simulated onboard the satellite using software.

B.1 Satellite Simulator Implementation

The satellite simulator is implemented using Java's inheritance so it is easy to create simulators for a particular satellite. Currently, satellite simulators have been created for both Cal Poly's CP6 and the University of Tokyo's XI-IV. These simulators mimic as much functionality as required to generate behavior which appears externally to be equivalent to the real satellite. This includes beacon

functionality and randomly generated satellite data.

B.2 Satellite Link Quality

To simulate a real space link between ASOF and the satellite, a connection model was implemented. This connection model allows an instantiation of the satellite simulator to specify both the rate of dropped packets and the bit error rate. In this way, one can test how ASOF reacts to poor link quality.

B.3 Responses File

In some instances, defining an explicit set of satellite responses is beneficial. For example, when defining a specific test situation, it is easier to list the satellite responses in a file as oppose to manually setting parameters in the satellite model. This functionality is created using a `ResponseSat` which takes as input a response file. The specific file format can be found in Appendix C.5.

Appendix C

File Formats

The following file formats are used throughout the ASOF framework.

C.1 MoredBs Log File Format

Instead of always using MoredBs' MySql database directly, a MoredBs log file format has been defined. Each pass is separated by a blank line and contains a number of operations events, one per line.

```
((<UP | DOWN>,<hex satellite data> \n)* \n)*
```

C.2 MXML File Format

A Mining XML (MXML) file is an XML file which specifies instances of a process. The complete file format can be found in the MXML XSD [50]. The following is the subset of the XSD tags used in the ASOF framework.

```
<WorkflowLog>
```

```
<Process id="0" description="">
<ProcessInstance id="" description="">
<AuditTrailEntry>
<WorkflowModelElement>...</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
</Process>
<Process>
...
</Process>
.
.
.
</WorkflowLog>
```

C.3 DFA File Format

The DFA file format encodes all the necessary information to recreate a DFA process model. The file format is as follows.

```
(<State Number>,<Agent Action> \n)*

(<From State Number>,<To State Number>,<Satellite Action> \n)*
```

C.4 Configuration File Formats

The following parameters are used in the ASOF framework's configuration files.

C.4.1 asof.prop

These properties relate to general ASOF operations.

- **mainSat:** The name of the satellite that this ASOF instance operates.
- **downImage:** The path of the downlink image.
- **upImage:** The path of the uplink image.
- **timeBetweenCommands:** The time to wait between sending commands.
- **defaultDataFormat:** Sets the default data format for uplinked and downlinked data. The available values are
  - dec
  - hex
  - ascii
- **defaultShowTimestamp:** The default setting to show timestamp information for all uplinked and downlinked data.

C.4.2 satellite.prop

The following properties relate to the specific satellite being operated.

- **lose:** The fully qualified package name of the Line of Sight Executive.
- **knowledgebase:** The fully qualified package name of the Knowledge Base.
- **dfaProcessModelLibPath:** Used for the DFA process model Knowledge Base, the path to the DFA process model files.
- **taskfile:** The path to the task file used for operations.
- **defaultTaskFile:** The path to the task file which will be executed after the main task file is completed.

- **tnc:** The fully qualified package name of the TNC.
- **tncHost:** Used for the TCP socket TNC, this is the host name of the server running the Satellite Simulator.
- **tncPort:** Used for the TCP socket TNC, this is the port number on which to connect to the Satellite Simulator.
- **tncResponseFile:** Used for the ResponseSat TNC, specifies the response file path to use as input.
- **tncCommPortUp:** Used for the serial TNC, specifies the comm port to use for uplinks.
- **tncCommPortDown:** Used for the serial TNC, specifies the comm port to use for downlinks.
- **tncCommPort:** Used for the serial TNC, specifies the uplink and downlink comm port if they are the same.

C.5 Satellite Simulator Response File

A set of responses can be defined for the satellite simulator to use by creating a file of the following format.

```
(((DataResponse | SatAction) \n)* \n)*
```

Each satellite response is on its own line and a blank line separates response sets.

Appendix D

Petri Nets Background

Petri nets are used as an intermediate data structure during DFA process model creation. Petri nets are bipartite graphs made up of Place and Transition nodes [32]. Places and Transitions can be connected together using directed arcs but a Place can never connect to another Place and a Transition can never connect to another Transition. A Place can have a number of tokens located in it at any time and a Transition can fire if all of its input Place nodes have tokens. Once a Transition fires, the tokens are moved from its input Place nodes to its output Place nodes. Petri nets have been used to model processes and resources in distributed environments. The formal definition can be found below.

Petri nets: A Petri net is a 5-tuple, PN = (P,T,F,W,M $_0$) where:

- P = $\{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
- T = $\{t_1, t_2, \cdots, t_m\}$ is a finite set of transitions,
- F $\subseteq$ (P $\times$ T) $\cup$ (T $\times$ P) is a set of arcs (flow relation),

- W: F $\rightarrow \{1, 2, 3, \cdots\}$ is a weight function,
- M_0: P $\rightarrow \{1, 2, 3, \cdots\}$ is the initial marking,
- P $\cap$ T $= \emptyset$ and P $\cup$ T $\neq \emptyset$

Petri nets can represent a number of relationships [48] between transitions (see Figure D.1 for a graphical representation of these relationships) which are

1. **Follows (A < B)**: Task B occurs after Task A
2. **Causal (A $\rightarrow$ B)**: Task B is always proceeded by Task A
3. **Parallel (A || B)**: Task A and Task B are done in parallel
4. **Unrelated (A # B)**: Task A occurs independently of Task B

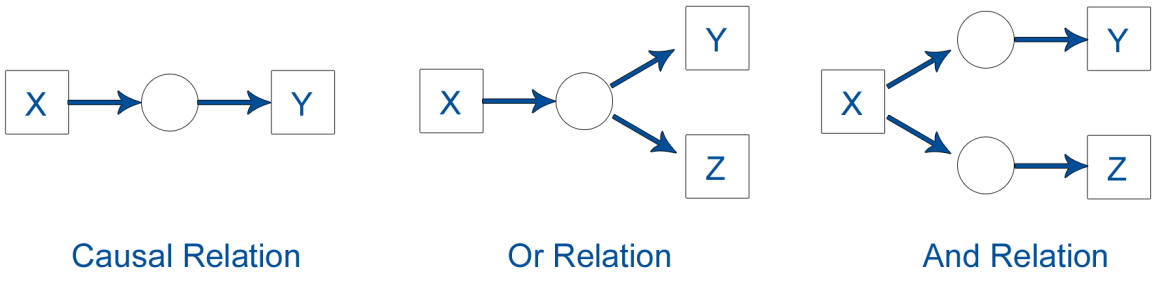


Figure D.1: Relationships Between Transitions

D.1 Workflow Nets

Workflow nets (WF-nets) are a subset of Petri nets and are more closely related to a DFA process model (see Figure D.2 for an example workflow net) [49]. Formally, a WF-net is N = (P, T, F), a Place-Transition net (P/T net) and $\bar{t}$ a fresh identifier not in P ∪ T. N must fulfill the following requirements:

1. Object creation: P contains an input place i such that $\bullet i = \emptyset$,
2. Object completion: P contains an output place o such that $o\bullet = \emptyset$,
3. Connectedness: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected,

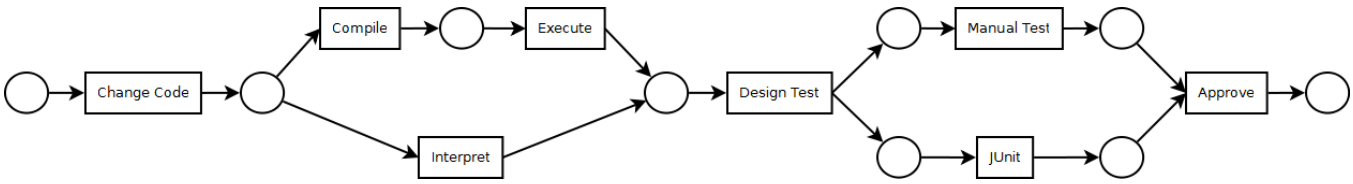


Figure D.2: An Example Workflow Net

Appendix E

The α-Algorithm

The following is the α-algorithm used during DFA process model creation [5].

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$

T_W is the set of all the distinct events which occur in the workflow log. These will be used to define all of the transitions in the resulting WF-net.

2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = first(\sigma)\}$

T_I is the set of all the possible ways for a process to start. This set will be used to create the connections from the starting place node.

3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = last(\sigma)\}$

T_O is the set of all the possible ways for a process to end. This set will be used to create the connections going to the ending place node.

4. $X_W = \{(A, B) \mid A \subseteq T_W \ \wedge \ B \subseteq T_W \ \wedge \ \forall_{a \in A} \forall_{b \in B} \ a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} \ a_1 \#_W a_2 \ \wedge \ \forall_{b_1, b_2 \in B} \ b_1 \#_W b_2\}$

X_W is the set of tuples (A, B) such that A causes B. That is for every B in our workflow log, it is always preceded by at least one A.

5. $Y_W = \{(A, B) \in X_W \mid \forall_{(A', B') \in X_W} \ A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$

Y_W is a subset of X_W, (A, B), such that A occurs immediately before B in at least one of the workflow log cases.

6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y_W\} \cup \{i_W, o_W\}$

P_W is the set of places connecting the transitions. They are constructed using the set of directly connected transitions defined in Y_W such that for every pair of transitions in Y_W, (A, B) there is a place between them defined as $P_{(A,B)}$.

7. $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\} \cup \{(p_{(A,B),b}) \mid (A, B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in TI\} \cup \{(t, o_W) \mid t \in TO\}$

F_W is the set of directed arcs in the resulting WF-net. The arcs are the connections from places to transitions and transitions to places using the previously created sets T_W and P_W. F_W contains the directed arcs (A, $P_{(A,B)}$) and ($P_{(A,B)}$, B) for every 'A' and 'B' from T_W and $P_{(A,B)}$ from P_W.

8. $\alpha(W) = (P_W, T_W, F_W)$

Using the sets created in previous steps(P_W, T_W, F_W) create the triple that is the WF-net.

E.1 α-Algorithm Example

The following is an example of the α-Algorithm in action.

Case #	Process Event
1	A
1	B
2	A
1	C
1	D
2	C
3	A
3	B
2	B
2	D
4	E
3	C
3	D
4	F

Table E.1: An Example Workflow Log

Case 1
A
B
C
D

Case 2
A
C
B
D

Case 3
A
B
C
D

Case 4
E
F

Table E.2: Organized Cases from the Example Workflow Log

1. $T_W = \{A, B, C, D, E, F\}$
2. $T_I = \{A, E\}$
3. $T_O = \{D, F\}$
4. $X_W = \{(A, B), (A, C), (A, D), (B, D), (C, D), (E, F)\}$

NOTE: (B, C) and (C, B) are not included in X_W since the causual relationship is not true.

5. $Y_W = \{(A, B), (A, C), (B, D), (C, D), (E, F)\}$

NOTE: Only (A, D) is in X_W and not in Y_W since there is no case where a 'D' is directly followed by an 'A'.

6. $P_W = \{P_{(A,B)}, P_{(A,C)}, P_{(B,D)}, P_{(C,D)}, P_{(E,F)}\}$
7. F_W = { (A, $P_{(A,B)}$), ($P_{(A,B)}$, B), (A, $P_{(A,C)}$), ($P_{(A,C)}$, C), (B, $P_{(B,D)}$), ($P_{(B,D)}$, D), (C, $P_{(C,D)}$), ($P_{(C,D)}$, D), (E, $P_{(E,F)}$), ($P_{(E,F)}$, F) }
8. $\alpha(W) = (P_W, T_W, F_W)$

The resulting WF-net is shown in Figure E.1.

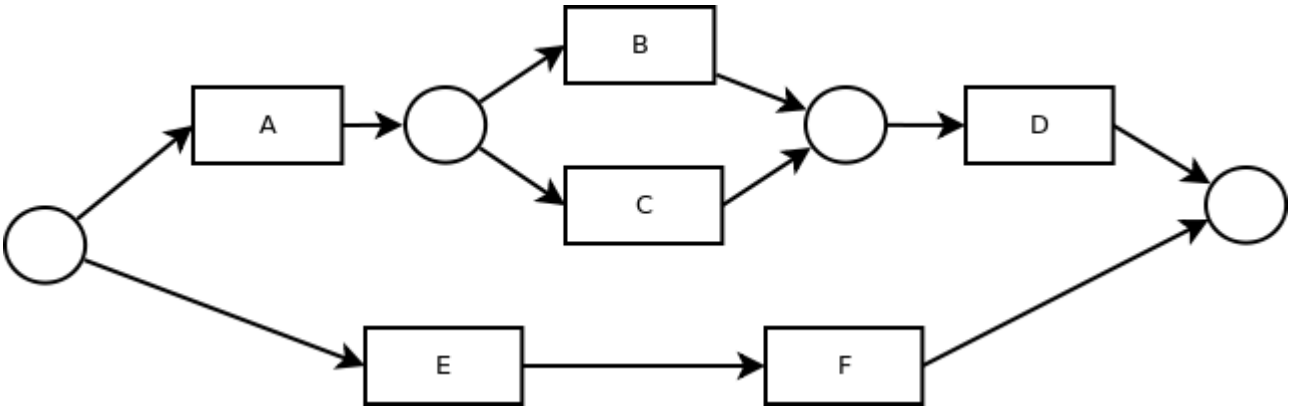


Figure E.1: The Completed Workflow Net Generated

E.2 α-Algorithm Assumption

In order for the alpha algorithm to work correctly, we must assume that the workflow log is complete. That is we must assume that all possible events and paths are present in the log. This assumption is made since if an event is not present in the log, then it cannot be included in the generated WF-net.

E.3 α-Algorithm Limitation

One of the greatest drawbacks of this version of the α-algorithm is that it does not support cycles of length 1. That is an event can never be repeated directly after it have been completed as is represented in Figure E.2.

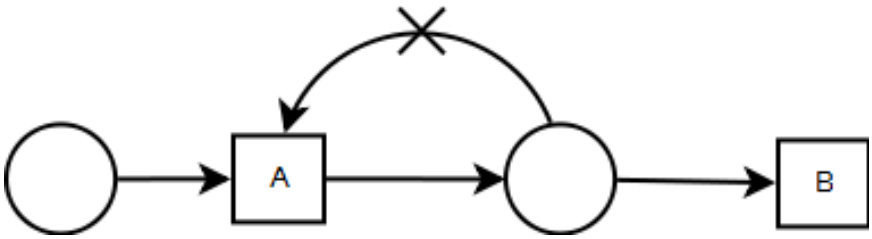


Figure E.2: No Single Loops Possible with the Basic α-Algorithm